35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2015, December 16–18, 2015, Bangalore, India

Edited by

Prahladh Harsha

G. Ramalingam

Editors

Prahladh Harsha
Tata Institute of Fundamental Research
Mumbai 400005
India
prahladh@tifr.res.in

G. Ramalingam
Microsoft Research India
Bangalore 560001
India
grama@microsoft.com

ACM Classification 1998
D.2.4 Software/Program Verification, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.3 Formal Languages

ISBN 978-3-939897-97-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-97-2.

Publication date
December, 2015

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2015.i

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 2A

Session 2B

Session 3A

Session 3B

Session 4A

Session 4B

Session 5A

Session 5B

Session 6A

Session 6B

Preface

The 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015), organized annually by the Indian Association for Research in Computing Science (IARCS), was held at the Indian Institute of Science, Bangalore, from December 16 to December 18, 2015.

The program consisted of 6 invited talks and 42 contributed papers. This proceedings volume contains the contributed papers and abstracts of invited talks presented at the conference. The proceedings of FSTTCS 2015 is published as a volume in the LIPIcs series under a Creative Commons license, with free online access to all, and with authors retaining rights over their contributions.

The 42 contributed papers were selected from a total of 117 submissions. We thank the program committee for its efforts in carefully evaluating and making these selections. We thank all those who submitted their papers to FSTTCS 2015. We also thank the external reviewers for sending their informative and timely reviews.

We are particularly grateful to the invited speakers: Boaz Barak (Harvard University & Microsoft Research), Ahmed Bouajjani (LIAFA, CNRS & Univ. Paris Diderot), Moses Charikar (Stanford University), Suresh Jagannathan (Purdue University), Ankur Moitra (MIT), and James Worrell (University of Oxford) who readily accepted our invitation to speak at the conference.

There were two pre-conference workshops, *Clustering Theory and Practice* (CTAP) and the *17th International Workshop on Verification of Infinite State Systems* (INFINITY 2015) and two post-conference workshops, *Algorithmic Verification of Real-Time Systems* (AVeRTS) and *Applications of Fourier Analysis to Theoretical Computer Science* (FOURIER). We thank Arnab Bhattacharyya (IISc Bangalore), Aiswarya Cyriac (Uppsala University), Amit Deshpande (Microsoft Research), Frédéric Herbreteau (Univ. Bordeaux, LaBRI), Ravishankar Krishnaswamy (Microsoft Research), M. Praveen (Chennai Mathematical Institute), and Krishna S. (IIT Bombay) for organizing these workshops.

On the administrative side, we thank the organizing committee led by Prof. Aditya Kanade (IISc Bangalore) and Prof. Deepak D'Souza (IISc Bangalore), who put in many months of effort in ensuring excellent conference and workshop arrangements at the Indian Institute of Science.

We would also like to thank Madhavan Mukund, Venkatesh Raman, and S. P. Suresh for promptly responding to our numerous questions and requests relating to the organization of the conference. We also thank the Easychair team whose software has made it very convenient to do many conference related tasks. Finally, we thank the Dagstuhl LIPIcs staff for their coordination in the production of this proceedings, particularly Marc Herbstritt who was very prompt and helpful in answering our questions.

Prahladh Harsha and G. Ramalingam
December 2015

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Programme Chairs

Prahladh Harsha (TIFR)
G. Ramalingam (Microsoft Research)

Programme Committee

Andrej Bogdanov (The Chinese University of Hong Kong)
Amit Deshpande (Microsoft Research)
Fedor Fomin (Univ. Bergen)
Naveen Garg (IIT Delhi)
Sariel Har-Peled (Univ. Illinois, Urbana-Champaign)
Nutan Limaye (IIT Bombay)
Meena Mahajan (IMSc)
Ruta Mehta (Georgia Tech.)
Alantha Newman (CNRS-Univ. Grenoble Alpes & G-SCOP)
Debmalya Panigrahi (Duke University)
Prasad Raghavendra (Univ. California, Berkeley)
Ramprasad Saptharishi (Tel Aviv University)
Pranab Sen (TIFR)
Suresh Venkatasubramanian (Univ. Utah)
Magnus Wahlstrom (Royal Holloway, Univ. London)
S. Akshay (IIT Bombay)
Parosh Abdulla (Uppsala University)
Erika Abraham (RWTH Aachen University)
Franck Cassez (Macquarie University)
Avik Chaudhuri (Facebook)
Thomas Colcombet (LIAFA, CNRS & Univ. Paris Diderot)
Stephanie Delaune (LSV, CNRS & ENS Cachan)
Javier Esparza (TU Munich)
Ashutosh Gupta (TIFR)
Ranjit Jhala (Univ. California, San Diego)
Roland Meyer (Univ. Kaiserslautern)
V. Krishna Nandivada (IIT Madras)
R. Ramanujam (IMSc)
Sriram Sankaranarayanan (Univ. Colorado Boulder)
Nishant Sinha (IBM Research)
S. P. Suresh (Chennai Mathematical Institute)

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organizing Committee

Deepak D'Souza (IISc Bangalore), co-chair
Rahul Gupta (IISc Bangalore)
Inzemamul Haque (IISc Bangalore)
Sabuj Kumar Jena (IISc Bangalore)
Shalini Kaleeswaran (IISc Bangalore)
Aditya Kanade (IISc Bangalore), co-chair
Pallavi Maiya (IISc Bangalore)
Suvam Mukherjee (IISc Bangalore)
Anirudh Santhiar (IISc Bangalore)

External Reviewers

Adsul, Bharat
Aggarwal, Divesh
Aiswarya, C.
Ambainis, Andris
Batmalle, Hadrien
Bhaskar, Umang
Bhattachar, Sayan
Brazdil, Tomas
Brenguier, Romain
Brihaye, Thomas
Broadbent, Christopher
Brotherston, James
Cadilhac, Michaël
Castro, Pablo
Chakarov, Aleksandar
Chakraborty, Souymodip
Chini, Peter
Chiplunkar, Ashish
Chistikov, Dmitry
Chitnis, Rajesh
Cormode, Graham
Cryan, Mary
Curticapean, Radu
D'Osualdo, Emanuele
D'Souza, Deepak
Delzanno, Giorgio
Elberfeld, Michael
Escoffier, Bruno
Forbes, Michael A.
Forejt, Vojtech
Fox, Kyle
Francis, Mathew
Freeman, Rupert
Furbach, Florian
Gairing, Martin
Gaspers, Serge
Ge, Rong
Ghica, Dan
Gibson, Matt
Golovach, Petr
Gujar, Sujit
Gupta, Manoj
Gyori, Benjamin
Höfner, Peter
Hague, Matthew
Haney, Samuel
Hoffmann, Philipp
Hofmann, Martin
Holik, Lukas
Horn, Florian
Im, Sungjin
Jain, Rahul
Jones, Mark
Karandikar, Prateek
Karmarkar, Hrishikesh
Kayal, Neeraj
Kell, Nathaniel
Kratsch, Stefan
Kretinsky, Jan
Krishnaswamy, Ravishankar
Kufleitner, Manfred
Kuich, Werner
Kulkarni, Janardhan
Kumar, Mrinal
Kupferman, Orna
Lagerkvist, Victor
Lengal, Ondrej
Lin, Anthony Widjaja
Lodaya, Kamal
Lohrey, Markus
Luttenberger, Michael
Manuel, Amaldev
Mayr, Richard
McGregor, Andrew
Mckenzie, Pierre
Melliès, Paul-André
Misra, Neeldhara
Mnich, Matthias
Mogavero, Fabio
Mohammad, Meesum Syed
Monmege, Benjamin
Mukhopadhyay, Partha
Mukund, Madhavan
Muscholl, Anca
Muskalla, Sebastian
Nasre, Meghana
Natarajan, Raja
Nimbhorkar, Prajakta

Norman, Gethin
Nyman, Ulrik
Otop, Jan
Panageas, Ioannis
Panolan, Fahad
Paperman, Charles
Park, Sungwoo
Parrow, Joachim
Paul, Soumya
Pavan, A
Phawade, Ramchandra
Pilipczuk, Marcin
Pilipczuk, Micha
Praveen, M.
Raman, Venkatesh
Reidl, Felix
Roy, Sambuddha
S., Krishna
Sabharwal, Yogish
Sanchez, Cesar
Satti, Srinivasa Rao
Saurabh, Nitin
Saurabh, Saket
Schewe, Sven
Schmitz, Sylvain
Seth, Anil
Shah, Simoni
Simon, Sunil Easaw
Soltys, Karolina
Sproston, Jeremy
Sreejith, A V
Srinivasan, Srikanth
Srivathsan, B
Streicher, Thomas
Sundararajan, Vaishnavi
Swamy, Chaitanya
Vishnoi, Nisheeth
Weidner, Thomas
Wilde, Mark
Wojtczak, Dominik
Yazdanbod, Sadra
Zetzsche, Georg

Bypassing Worst Case Analysis: Tensor Decomposition and Clustering

Moses S. Charikar

Computer Science Department, Stanford University
Stanford, CA, USA
moses@cs.stanford.edu

Abstract

Typical worst case analysis of algorithms has led to a rich theory, but suffers from many pitfalls. This has inspired several approaches to bypass worst case analysis. In this talk, I will describe two vignettes from recent work in this realm.

In the first part of the talk, I will discuss tensor decomposition – a natural higher dimensional analog of matrix decomposition. Low rank tensor decompositions have proved to be a powerful tool for learning generative models, and uniqueness results give them a significant advantage over matrix decomposition methods. Yet, they pose significant challenges for algorithm design as most problems about tensors are NP-hard. I will discuss a smoothed analysis framework for analyzing algorithms for tensor decomposition which models realistic instances of learning problems and allows us to overcome many of the usual limitations of using tensor methods.

In the second part of the talk, I will explore the phenomenon of convex relaxations returning integer solutions. Clearly this is not true in the worst case. I will discuss instances of discrete optimization problems where, for a suitable distribution on inputs, LP and SDP relaxations produce integer solutions with high probability. This has been studied in the context of LP decoding, sparse recovery, stochastic block models and so on. I will mention some recent results for clustering problems: when points are drawn from a distribution over k sufficiently separated clusters, the well known k-median relaxation and a (not so well known) SDP relaxation for k-means exactly recover the clusters.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases tensor decomposition, smoothed analysis, convex relaxations, integrality

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.1

Category Invited Talk

35th IARCS Annual Conf. Foundations of Software Technology Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 1–1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Checking Correctness of Concurrent Objects: Tractable Reductions to Reachability

Ahmed Bouajjani[1], Michael Emmi[2], Constantin Enea[3], and Jad Hamza[3]

1 **Université Paris Diderot and Institut Universitaire de France, Paris, France**
abou@liafa.univ-paris-diderot.fr
2 **IMDEA Software Institute, Madrid, Spain**
michael.emmi@imdea.org
3 **Université Paris Diderot, Paris, France**
{cenea,jad.hamza}@liafa.univ-paris-diderot.fr

Abstract

Efficient implementations of concurrent objects such as semaphores, locks, and atomic collections including stacks and queues are vital to modern computer systems. Programming them is however error prone. To minimize synchronization overhead between concurrent object-method invocations, implementors avoid blocking operations like lock acquisition, allowing methods to execute concurrently. However, concurrency risks unintended inter-operation interference. Their correctness is captured by *observational refinement* which ensures conformance to atomic reference implementations. Formally, given two libraries L_1 and L_2 implementing the methods of some concurrent object, we say L_1 *refines* L_2 if and only if every computation of every program using L_1 would also be possible were L_2 used instead.

Linearizability [11], being an equivalent property [8, 5], is the predominant proof technique for establishing observational refinement: one shows that each concurrent execution has a linearization which is a valid sequential execution according to a specification, given by an abstract data type or atomic reference implementation.

However, checking linearizability is intrinsically hard. Indeed, even in the case where method implementations are finite-state and object specifications are also finite-state, and when a fixed number of threads (invoking methods in parallel) is considered, the linearizability problem is EXPSPACE-complete [9], and it becomes undecidable when the number of threads is unbounded [3]. These results show in particular that there is a complexity/decidability gap between the problem of checking linearizability and the problem of checking reachability (i.e., the dual of checking safety/invariance properties), the latter being, PSPACE-complete and EXPSPACE-complete in the above considered cases, respectively.

We address here the issue of investigating cases where tractable reductions of the observational refinement/linearizability problem to the reachability problem, or dually to invariant checking, are possible. Our aim is (1) to develop algorithmic approaches that avoid a systematic exploration of all possible linearizations of all computations, (2) to exploit existing techniques and tools for efficient invariant checking to check observational refinement, and (3) to establish decidability and complexity results for significant classes of concurrent objects and data structures.

We present two approaches that we have proposed recently. The first approach [5] introduces a parameterized approximation schema for detecting observational refinement violations. This approach exploits a fundamental property of shared-memory library executions: their histories are *interval orders*, a special case of partial orders which admit *canonical representations* in which each operation o is mapped to a positive-integer-bounded interval $I(o)$. Interval orders are equipped with a natural notion of *length*, which corresponds to the smallest integer constant for which an interval mapping exists. Then, we define a notion of bounded-interval-length analysis, and demonstrate its efficiency, in terms of complexity, coverage, and scalability, for detecting observational refinement bugs.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 2–4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The second approach [4] focuses on a specific class of abstract data types, including common concurrent objects and data structures such as stacks and queues. We show that for this class of objects, the linearizability problem is actually as hard as the control-state reachability problem. Indeed, we prove that in this case, the existence of linearizability violations (i.e., finite computations that are not linearizable), can be captured *completely* by a finite number of finite-state automata, even when an unbounded number of parallel operations is allowed (assuming that libraries are data-independent).

Related work. Several semi-automated approaches for proving linearizability, and thus observational refinement, have relied on annotating operation bodies with *linearization points* [2, 12, 13, 15, 16], to reduce the otherwise-exponential space of possible history linearizations to one single linearization. These methods often rely on programmer annotation, and do not admit conclusive evidence of a violation in the case of a failed proof.

Existing automated methods for proving linearizability of an atomic object implementation are also based on reductions to safety verification [1, 10, 15]. Abdulla et al. [1] is and Vafeiadis [15] consider implementations where operation's *linearization points* are fixed to particular source-code locations. Such approaches are incomplete since not all implementations have fixed linearization points (see for instance [7]). Aspect-oriented proofs [10] reduce linearizability to the verification of four simpler safety properties. However, this approach has only been applied to queues, and has not produced a fully automated and complete proof technique. Dodds et al. [7] prove linearizability of stack implementations with an automated proof assistant. Their approach does not lead to full automation however, e.g., by reduction to safety verification.

Automated approaches for detecting linearizability violations such as Line-Up [6] enumerate the exponentially-many possible history linearizations. This exponential cost effectively limits such approaches to executions with few operations. Colt [14]'s approach mitigates this cost with programmer-annotated linearization points, as in the previously-mentioned approaches, and ultimately suffers from the same problem: a failed proof only indicates incorrect annotation.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.2.5 Testing and Debugging

Keywords and phrases Concurrent objects, Linearizability, Verification, Bug detection

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.2

Category Invited Talk

References

1 Parosh Aziz Abdulla, Frédéric Haziza, Lukás Holík, Bengt Jonsson, and Ahmed Rezine. An integrated specification and verification technique for highly concurrent data structures. In *TACAS*, pages 324–338, 2013.

2 Daphna Amit, Noam Rinetzky, Thomas W. Reps, Mooly Sagiv, and Eran Yahav. Comparison under abstraction for verifying linearizability. In *CAV'07*, volume 4590 of *LNCS*, pages 477–490, 2007.

3 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. Verifying concurrent programs against sequential specifications. In *ESOP'13*. Springer, 2013.

4 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. On reducing linearizability to state reachability. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2015.

5 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. Tractable refinement checking for concurrent objects. In *POPL'15*. ACM, 2015.

6 Sebastian Burckhardt, Chris Dern, Madanlal Musuvathi, and Roy Tan. Line-Up: a complete and automatic linearizability checker. In *PLDI'10*, pages 330–340. ACM, 2010.

7 Mike Dodds, Andreas Haas, and Christoph M. Kirsch. A scalable, correct time-stamped stack. In *POPL'15*. ACM, 2015.

8 Ivana Filipovic, Peter W. O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *Theor. Comput. Sci.*, 411(51-52):4379–4398, 2010.

9 Jad Hamza. On the complexity of linearizability. In *3rd Intern. Conf. on Networked Systems, NETYS'15, Agadir, Morocco*, volume 9466 of *Lecture Notes in Computer Science*. Springer, 2015.

10 Thomas A. Henzinger, Ali Sezgin, and Viktor Vafeiadis. Aspect-oriented linearizability proofs. In *CONCUR*, pages 242–256, 2013.

11 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

12 Yang Liu, Wei Chen, Yanhong A. Liu, and Jun Sun. Model checking linearizability via refinement. In *FM'09*, volume 5850 of *LNCS*, pages 321–337, 2009.

13 Peter W. O'Hearn, Noam Rinetzky, Martin T. Vechev, Eran Yahav, and Greta Yorsh. Verifying linearizability with hindsight. In *PODC'10*, pages 85–94. ACM, 2010.

14 Ohad Shacham, Nathan Grasso Bronson, Alex Aiken, Mooly Sagiv, Martin T. Vechev, and Eran Yahav. Testing atomicity of composed concurrent operations. In *OOPSLA'11*, pages 51–64. ACM, 2011.

15 Viktor Vafeiadis. Automatically proving linearizability. In *CAV'10*, volume 6174 of *LNCS*, pages 450–464, 2010.

16 Shao Jie Zhang. Scalable automatic linearizability checking. In *ICSE'11*, pages 1185–1187. ACM, 2011.

Reachability Problems for Continuous Linear Dynamical Systems*

James Worrell

Department of Computer Science, University of Oxford
Parks Road, Oxford OX1 3QD, UK
james.worrell@cs.ox.ac.uk

Abstract

This talk is about reachability problems for continuous-time linear dynamical systems. A central decision problem in this area is the Continuous Skolem Problem [1], which asks to determine the existence of a zero of a real-valued function f satisfying an ordinary linear differential equation

$$f^{(n)} + a_{n-1}f^{(n-1)} + \ldots + a_0 f = 0$$

with coefficients $a_0, \ldots, a_{n-1} \in \mathbb{Q}$ and initial conditions $f(0), \ldots, f^{(n-1)}(0) \in \mathbb{Q}$. An alternative formulation of the problem asks whether the solution $x(t) \in \mathbb{R}^n$ of a given differential equation $x' = Ax + b$, with A a rational $n \times n$ matrix and b a rational n-dimensional vector, reaches a given halfspace.

The nomenclature *Continuous Skolem Problem* arises by analogy with the Skolem Problem for linear recurrence sequences [4]. The latter problem asks whether a sequence of integers satisfying a given linear recurrence has a zero term. Decidability is open for both the discrete and continuous versions of the Skolem Problem.

We show that the Continuous Skolem Problem lies at the heart of many natural computational problems on linear dynamical systems, such as reachability in continuous-time Markov chains and linear hybrid automata. We describe some recent work, done in collaboration with Chonev and Ouaknine [2, 3], that uses results in transcendence theory and real algebraic geometry to obtain decidability for certain variants of the problem. In particular, we consider a bounded version of the Continuous Skolem Problem, corresponding to time-bounded reachability. We prove decidability of the bounded problem assuming Schanuel's conjecture, a central conjecture in transcendence theory. We also describe some partial decidability results in the unbounded case in the case of functions f satisfying differential equations of fixed low order.

Finally, we give evidence of significant mathematical obstacles to proving decidability of the Continuous Skolem Problem in full generality by exhibiting some number-theoretic consequences of the existence of a decision procedure for this problem.

1998 ACM Subject Classification G.1.7 Ordinary Differential Equations

Keywords and phrases Linear Differential Equations, Continuous-Time Markov Chains, Hybrid Automata, Schanuel's Conjecture

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.5

Category Invited Talk

* This work was partially supported by the EPSRC.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 5–6

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

References

1 Paul C. Bell, Jean-Charles Delvenne, Raphaël M. Jungers, and Vincent D. Blondel. The Continuous Skolem-Pisot Problem. *Theoretical Computer Science*, 411(40-42):3625–3634, 2010.

2 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the decidability of the Bounded Continuous Skolem Problem. *CoRR*, abs/1506.00695, 2015.

3 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the decidability of the continuous infinite zeros problem. *CoRR*, abs/1507.03632, 2015.

4 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's Problem – on the border between decidability and undecidability. Technical Report 683, Turku Centre for Computer Science, 2005.

Convexity, Bayesianism, and the Quest Towards Optimal Algorithms

Boaz Barak

Harvard SEAS, Cambridge, MA, US, and Microsoft Research, Cambridge, MA, US
info@boazbarak.org

Abstract

In this high level and accessible talk I will describe a recent line of works aimed at trying to understand the intrinsic complexity of computational problems by finding *optimal* algorithms for large classes of such problems. In particular, I will talk about efforts centered on convex programming as a source for such candidate algorithms. As we will see, a byproduct of this effort is a computational analog of *Bayesian probability* that is of its own interest.

I will demonstrate the approach using the example of the *planted clique* (also known as *hidden clique*) problem – a central problem in average case complexity with connections to machine learning, community detection, compressed sensing, finding Nash equilibrium and more. While the complexity of the planted clique problem is still wide open, this line of works has led to interesting insights on it.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases Convex programming, Bayesian probability, Average-case complexity, Planted clique

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.7

Category Invited Talk

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 7–7

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Beyond Matrix Completion

Ankur Moitra

Department of Mathematics, MIT
Cambridge, MA, USA
moitra@mit.edu

Abstract

In this talk, we study some of the statistical and algorithmic problems that arise in recommendation systems. We will be interested in what happens when we move beyond the matrix setting, to work with higher order objects – namely, tensors. To what extent does inference over more complex objects yield better predictions, but at the expense of the running time? We will explore the computational vs. statistical tradeoffs for some basic problems about recovering approximately low rank tensors from few observations, and will show that our algorithms are nearly optimal among all polynomial time algorithms, under natural complexity-theoretic assumptions.

This is based on joint work with Boaz Barak.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases matrix completion, recommendation systems, tensor prediction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.8

Category Invited Talk

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 8–8

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Relational Refinement Types for Higher-Order Shape Transformers

Suresh Jagannathan

Department of Computer Science, Purdue University, IN, US
suresh@cs.purdue.edu

Abstract

Understanding, discovering, and proving useful properties of sophisticated data structures are central problems in program verification. A particularly challenging exercise for shape analyses involves reasoning about sophisticated shape transformers that preserve the shape of a data structure (*e.g.*, the data structure skeleton is always maintained as a balanced tree) or the relationship among values contained therein (*e.g.*, the *in-order* relation of the elements of a tree or the *parent-child* relation of the elements of a heap) across program transformations.

In this talk, we consider the specification and verification of such transformers for ML programs. The structural properties preserved by transformers can often be naturally expressed as inductively-defined *relations* over the recursive structure evident in the definitions of the datatypes they manipulate. By carefully augmenting a refinement type system with support for reasoning about structural relations over algebraic datatypes, we realize an expressive yet decidable specification language, capable of capturing useful structural invariants, which can nonetheless be automatically verified using off-the-shelf type checkers and theorem provers. Notably, our technique generalizes to definitions of *parametric* relations for polymorphic data types which, in turn, lead to highly composable specifications over higher-order polymorphic shape transformers.

1998 ACM Subject Classification D.2.4 Software/Program Verification-Correctness proofs, Formal Methods, D.3.2 Applicative (Functional) Languages, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Relational Specifications; Inductive and Parametric Relations; Refinement Types, Shape Analysis, Data Structure Verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.9

Category Invited Talk

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 9–9

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Robust Reoptimization of Steiner Trees*

Keshav Goyal[1] and Tobias Mömke[2]

1 **IIT Delhi, India,** mt5100599@maths.iitd.ac.in
2 **Saarland University, Germany,** moemke@cs.uni-saarland.de

Abstract

In reoptimization problems, one is given an optimal solution to a problem instance and a local modification of the instance. The goal is to obtain a solution for the modified instance. The additional information about the instance provided by the given solution plays a central role: we aim to use that information in order to obtain better solutions than we are able to compute from scratch.

In this paper, we consider Steiner tree reoptimization and address the optimality requirement of the provided solution. Instead of assuming that we are provided an optimal solution, we relax the assumption to the more realistic scenario where we are given an approximate solution with an upper bound on its performance guarantee.

We show that for Steiner tree reoptimization there is a clear separation between local modifications where optimality is crucial for obtaining improved approximations and those instances where approximate solutions are acceptable starting points. For some of the local modifications that have been considered in previous research, we show that for every fixed $\epsilon > 0$, approximating the reoptimization problem with respect to a given $(1+\epsilon)$-approximation is as hard as approximating the Steiner tree problem itself (whereas with a given optimal solution to the original problem it is known that one can obtain considerably improved results). Furthermore, we provide a new algorithmic technique that, with some further insights, allows us to obtain improved performance guarantees for Steiner tree reoptimization with respect to all remaining local modifications that have been considered in the literature: a required node of degree more than one becomes a Steiner node; a Steiner node becomes a required node; the cost of one edge is increased.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases reoptimization, approximation algorithms, Steiner tree problem, robustness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.10

1 Introduction

The Steiner tree problem (STP) is one of the most studied problems in the area of network design. We are given a graph G with nodes $V(G)$, edges $E(G)$, and a cost function $c\colon E(G) \to \mathbb{R}_{\geq 0}$, as well as a set $R \subseteq V(G)$ of required nodes (also called regular nodes or terminals). The objective is to find a minimum cost tree T within G such that $R \subseteq V(T)$. The Steiner tree problem is known to be APX-hard [8], and the currently best approximation algorithm has a performance guarantee of $\ln 4 + \epsilon \approx 1.387$ [24].

* Research partially funded by Deutsche Forschungsgemeinschaft grant BL511/10-1 and by the Indo-German Max Planck Center for Computer Science (IMPECS).

We consider the Steiner tree problem with respect to reoptimization, a framework for dynamic algorithms in the context of NP-hard problems. We are given two related instances I and I' of an algorithmic problem together with a solution Sol to the instance I, and our goal is to compute a solution to I'. The relation between I and I' is determined by an operation that we call *local modification*.

The concept of reoptimization is motivated by the observation that instead of computing new solutions from scratch, oftentimes we can reuse the effort spent to solve problems similar to the one at hand. For instance, let us consider a large circuit where certain components have to be connected. The components are the required nodes and there are points that may be used by several connections, the Steiner nodes. Now suppose that a long and costly computation has led to an almost optimal solution. Afterwards the requirements change: either an additional component has to be placed to a point that previously was a Steiner node or a component is removed, which turns a required node into a Steiner node. In such a situation it would seem wasteful to discard the entire previous effort.

Classically, when considering reoptimization problems one assumes that Sol is an optimal solution. The reason for this assumption is that assuming optimality considerably reduces the formal overhead and therefore facilitates to concentrate on the main underlying properties of the reoptimization problem. We show, however, that assuming optimality is not without loss of generality. Let us assume that $c(\text{Sol})$ is a $(1+\epsilon)$ factor larger than the cost of an optimal solution. Then we say that a Steiner tree reoptimization algorithm is *robust*, if it is an approximation algorithm and its performance guarantee is $\alpha \cdot (1 + O(\epsilon))$, where α is its performance guarantee when $\epsilon = 0$. Intuitive, this definition ensures that for $\epsilon \to 0$, the performance guarantee converges smoothly towards α, independent of the given instance. We consider robustness of reoptimization algorithms to be a crucial feature, since in real world applications close to optimal solutions are much more frequent than optimal solutions.

We address all local modifications that have previously been considered for Steiner tree reoptimization. We classify these modifications into two groups, according to their robustness. The first group contains those problems where obtaining a robust reoptimization algorithm implies to provide an approximation algorithm for the (non-reoptimization) Steiner tree problem with matching performance guarantee. The second group of problems allows for improved robust reoptimization algorithms compared to STP approximation algorithms.

For all reoptimization problems of the second group that have previously been considered (and that are known to be NP-hard [15]), we provide robust reoptimization algorithms that, for $\epsilon \to 0$, obtain better performance guarantees than the previous results with optimality assumption [12, 13].

1.1 Local Modifications and Our Contribution

There are ten local modifications that previously have been considered for the Steiner tree problem. The two most studied modifications address the set of required nodes: we either declare a required node to be a Steiner node, or a Steiner node to be a required node. Here, STP^{R-} resp. STP^{R+} denote the corresponding reoptimization problems. We show, in Section 4, that finding a robust reoptimization algorithm for STP^{R-} is as hard as finding a Steiner tree approximation algorithm with matching approximation ratio. If one, however, excludes that the node t declared to be a Steiner node is a leaf in the given instance, we provide a robust reoptimization algorithm with improved performance ratio (see Table 1 for an overview of the achieved improvements). We show that in contrast to STP^{R-}, STP^{R+} always allows for improved robust reoptimization algorithms. The next interesting type of local modification is to modify the cost of a single edge. We do not require the cost function

to be metric. In particular, in the shortest path metric induced by the modified edge cost, the cost of several edges may be changed. We call the modification where the cost of one edge is increased STP^{E+}, and the converse local modification where the cost of one edge is decreased is STP^{E-}. We provide an improved robust reoptimization algorithm for STP^{E+} and show that robust reoptimization for STP^{E-} is as hard as approximating the Steiner tree problem itself (analogous to general STP^{R-}). The two local modifications to remove an edge from the graph and to add an edge to the graph reduce to STP^{E+} resp. STP^{E-} in a straightforward manner.

The remaining four local modifications are the removal or addition of a required node or a Steiner node. It is known that the local modification where required or Steiner nodes are removed is as hard as Steiner tree approximation, even if we are given an optimal solution to the old problem [15]. We show that adding a required node or a Steiner node to the graph causes robust reoptimization to be as hard as STP approximation.

One of the key insights that leads to our improved algorithms is that for all local modifications that allow for robust reoptimization algorithms, we can replace the given Steiner tree by a k-restricted Steiner tree of roughly the same cost. At the same time, we have the promise that there is an almost optimal Steiner tree for the modified instance that is k-restricted. This property allows us to handle certain subgraphs of Steiner trees called full components. (i) We remove entire full components from the given Steiner tree and perform optimal computations to obtain a feasible solutions to the modified instance, and (ii) we *guess* entire full components of the Steiner tree that we aim to compute. The new insights simplify and generalize the previous approaches to Steiner tree reoptimization and therefore give raise to more sophisticated analyses than before.[1]

Due to space constraints, we restrict the presentation to analyzing STP^{R-} and STP^{E+}, as these local modification give the best overview of the used techniques and ideas.

1.2 Related Work

The concept of reoptimization was first mentioned by Schäffter [29] in the context of postoptimality analysis for a scheduling problem. Since then, the concept of reoptimization has been investigated for several different problems, including the traveling salesman problem [1, 5, 14, 18, 7, 28], the rural postman problem [3], fast reoptimization of the spanning tree problem [23], the knapsack problem [2], covering problems [11], the shortest common superstring problem [10], maximum weight induced heredity problems [21], and scheduling [29, 6, 20]. There are several overviews on reoptimization [4, 22, 17, 31].

The Steiner tree reoptimization problem in general weighted graphs was previously investigated in [9, 26, 16, 15, 12, 13], see Table 1.

2 Preliminaries

We denote a Steiner tree instance by (G, R, c), where G is an undirected graph, $R \subseteq V(G)$ is the set of required nodes, and $c\colon E(G) \to \mathbb{R}_{\geq 0}$ is a cost function. The *Steiner nodes* of (G, R, c) are the nodes $S = V(G) \setminus R$.

Since c is symmetric, we sometimes use the simplified notation $c(u, v) = c(v, u)$ instead of $c(\{u, v\})$.

[1] We note that with some additional effort, it would also be possible to adapt the technique of Bilò and Zych [13] and use them for our results.

Table 1 Comparison of approximation ratios of the Steiner Tree Reoptimization problem for the different types of local modifications. To increase the readability, all values ϵ, δ in the approximation ratios are omitted. The numerical values are rounded up at the third digit and we assume $\beta = 1.387$, the approximation ratio $\ln(4) + \epsilon$ of the Steiner tree approximation algorithm of Byrka et al. [24] with small enough ϵ.

Local Modification	**Our Results** **Sol: $(1+\epsilon)$-Approx.**		**Previous Results** **Sol: Optimal Solution**	
	Expression	**Value**	**Expression**	**Value**
STP^{R-} (internal node)	$\frac{10\beta-7}{7\beta-4}$	1.204	$\frac{3\beta-2}{2\beta-1}$ [13]	1.219
STP^{R-} (leaf node)	not robust	1.204 if $\epsilon = 0$	$\frac{3\beta-2}{2\beta-1}$ [13]	1.219
STP^{R+}	$\frac{10\beta-7}{7\beta-4}$	1.204	$\frac{3\beta-2}{2\beta-1}$ [12]	1.219
STP^{E+}	$\frac{7\beta-4}{4\beta-1}$	1.256	$\frac{2\beta-1}{\beta}$ [13]	1.29
STP^{E-}	not robust	1.387	$\frac{5\beta-3}{3\beta-1}$ [9] assuming metricity	1.246
Add Node	not robust	1.387	without [24]: 1.5 [26]	1.387
Remove Node	not robust	1.387	As hard as STP approx. [15]	1.387

For two graphs G, G', we define $G \cup G'$ to be the graph with node set $V(G) \cup V(G')$ and edge set $E(G) \cup E(G')$ (i. e., we do not keep multiple edges). For an edge e, $G - e$ is G with e removed from $E(G)$. We define $G - G'$ to be the graph with node set $V(G) \setminus (V(G') \cap S)$ and edge set $E(G) \setminus E(G')$. We emphasize that we do *not* remove required vertices.

In Steiner tree algorithms, it is standard to consider the edge-costs to be metric. The reason is that forming the metric closure (i. e., using the shortest path metric) does not change the cost of an optimal solution: if we replacing an edge of a Steiner tree by the shortest path between the two ends, we obtain a valid Steiner tree again.

In the context of reoptimization, however, we cannot assume the cost function to be metric without loss of generality, because the triangle inequality restricts the effect of local changes. Therefore in the following we have to carefully distinguish between metric and general cost functions.

For a given Steiner tree, its *full components* are exactly those maximal subtrees that have all leaves are in R and all internal nodes are in S. Note that for a given Steiner tree T, we may remove leaves if they are not in R; we still have a Steiner tree, and its cost did not increase. Therefore we may assume that T is composed of full components. A *k-restricted Steiner tree* is a Steiner tree where each full component has at most k nodes from R.

▶ **Lemma 1** (Borchers, Du [19]). *For an arbitrary $\epsilon > 0$ there is a $k \in O_\epsilon(1)$ such that for all Steiner tree instances (G, R, c) with optimal solution* Opt *of cost* opt *and c is a metric, there is a k-restricted Steiner tree T of cost at most $(1+\epsilon)$*opt *which can be obtained from* Opt *in polynomial time.*

We assume that in k-restricted Steiner trees T where c is a metric, the Steiner nodes $v \in V(T) \cap S$ have a degree of $\deg(v) \geq 3$. This is without loss of generality, since $\deg(v) \geq 2$ by the definition of k-restricted Steiner trees; if $\deg(v) = 2$ and u, w are the neighbors of v, $c(u, v) + c(v, w) \geq c(u, w)$. We replace $\{u, v\}, \{v, w\}$ by $\{u, w\}$ without increasing the cost of T and without changing the property that T is k-restricted.

Input : A Steiner tree instance (G, R, c),
a Steiner forest F in G with trees $F_1, F_2, \ldots, F_\ell$
Output : A Steiner tree T
Set $G' := G/F$ such that F_i is contracted to v_i;
Set $R' := \{v_i : V(F_i) \cap R \neq \emptyset\}$;
Compute a minimum Steiner tree T' of (G', R', c);
Obtain T from T' by expanding F.

Algorithm 1: Connect

Within the entire text, Opt denotes an optimal solution and opt denotes the cost of an optimal solution. We will often add sub- and superscripts to Opt and opt in order to distinguish between various types of (close to) optimal solutions.

3 Connecting Forests and Guessing Components

We state two algorithms that we will use repeatedly within the subsequent sections. The first algorithm, Connect, was introduced by Böckenhauer et al. [16] and has been used in all previous Steiner tree reoptimization results. The algorithm connects components of a Steiner forest F of G in order to obtain a feasible Steiner tree T. The idea is that we start from a partial solution with few components that together contain all required vertices, and we use an exact computation to complete the solution. In Connect we use the following notation. Denote by G/V' for $V' \subseteq V(G)$ the contraction of V' in G. We write G/F instead of $G/V(F)$, if F is a subgraph of G. Note that after contracting a component there may be multiedges. Here, we treat multigraphs as simple graphs, where we only consider the cheapest edge of each multiedge. For ease of presentation, we slightly abuse notation and use the cost function c for both the graph before and the graph after the contraction.

Clearly, the graph T computed by Connect is a Steiner tree. If the number of components ℓ of the forest F given as input is a constant, by using the Dreyfuss-Wagner algorithm [25][2] to compute T', Connect runs in polynomial time. The graph T computed by Connect is the minimum cost Steiner tree that contains F, since all Steiner trees that contain F determine feasible solutions T'.

The second algorithm of this section, Guess, which is motivated from the CONNECT algorithm of [13] and presented here in a different manner, provides a mechanism to profit from guessing full components of an optimal k-restricted Steiner tree: we compress the guessed full components to single vertices and this way we obtain a new instance to which we apply known approximation algorithms. We call Guess by simply writing Guess(ℓ), if the instance and k are clear from the context and $\mathcal{A}$ is a β-approximation algorithm. Note that for instance Guess($3k$) means that $\ell = 3k$.

▶ **Lemma 2.** *For an arbitrary $\epsilon > 0$, let k be the parameter obtained from Lemma 1. Let $\mathcal{A}$ be a polynomial time β-approximation algorithm for the Steiner tree problem. Furthermore, let* Opt_k *be an optimal k-restricted solution of cost* opt_k *to the Steiner tree instance (G, R, c) where c is a metric. Then, for $\ell \in O_\epsilon(1)$,* Guess *runs in polynomial time and computes a Steiner tree T of cost at most $(1+\epsilon)(\beta - \beta\zeta + \zeta)\text{opt}$, where ζopt_k is the total cost of the ℓ maximum weight full components of* Opt_k *and* opt *is the cost of an optimal solution.*

[2] We refer to Hougardy et al. [27] for an overview of further exact Steiner tree algorithms that, depending on the given parameters, may be faster than Dreyfuss-Wagner.

Input : A Steiner tree instance (G, R, c) with c metric, numbers $\ell, k \in \mathbb{N}$, and a Steiner tree approximation algorithm $\mathcal{A}$
Output : A Steiner Tree T
Run $\mathcal{A}$ on (G, R, c) and obtain a Steiner tree T;
foreach $\mathcal{S} = \{S_1, S_2, \ldots, S_\ell\}$ *such that* $S_i \subseteq V$ *with* $|S_i| \leq 2k$ *and* $2 \leq |S_i \cap R| \leq k$ *for* $1 \leq i \leq \ell$ **do**
  For each i, compute a minimum spanning tree T_i with $V(T_i) = S_i$;
  Contract each T_i to a required node r_i;
  Run $\mathcal{A}$ on the resulting instance;
  Obtain T' by expanding the contracted components of each r_i;
  if $c(T') < c(T)$ **then**
    Replace T by T'.

Algorithm 2: GUESS

Proof. We first analyze the running time of the algorithm. Since $\mathcal{A}$ runs in polynomial time, we only have to consider the number of families $\mathcal{S}$ that we have to test. This number is bounded from above by $\left(\sum_{i=2}^{2k} \binom{n}{i}\right)^\ell$, since we only choose sets of size at most $2k$. Since both k and ℓ are constants, this number is polynomial in n.

Next we analyze the cost of T. Since we assume that for each Steiner node $v \in S \cap V(\text{OPT}_k)$, $\deg(v) \geq 3$, we conclude that all full components of OPT_k have at most $2k$ nodes. Therefore there is a family $\mathcal{S}$ considered by GUESS such that the classes of $\mathcal{S}$ are exactly the node sets of the ℓ maximum weight full components of OPT_k. Contracting a minimum spanning tree T_i is equivalent to contracting the full component with required nodes $R \cap S_i$ in OPT_k. We finish the proof by applying a standard argument that was used, for instance, by Böckenhauer et al. [14]. The cost of an optimal Steiner tree before expanding the full components is bounded from above by $\text{opt}_k - \zeta\text{opt}_k$, and expanding the full components adds ζopt_k. Therefore we obtain $c(T) \leq \beta(\text{opt}_k - \zeta\text{opt}_k) + \zeta\text{opt}_k = (\beta - \beta\zeta + \zeta)\text{opt}_k$. By our choice of k and Lemma 1, $\text{opt}_k \leq (1+\epsilon)\text{opt}$ and therefore $c(T) \leq (1+\epsilon)(\beta - \beta\zeta + \zeta)\text{opt}$. ◀

In the subsequent proofs, we will repeatedly obtain a value η such that $\zeta \geq (\alpha - 1 - \epsilon)\eta$, where α is the actual performance ratio of the considered approximation algorithm. By simple arithmetics and assuming that $(1+\epsilon)(\beta - \beta\zeta + \zeta)$ tends to $(\beta - \beta\zeta + \zeta)$ for ϵ chosen sufficiently small, Lemma 2 implies

$$\alpha \leq \frac{\beta + \beta\eta - \eta + \epsilon(\beta\eta - \eta)}{1 + \beta\eta - \eta}. \tag{1}$$

The reason for our assumption is that we can choose k in Lemma 1 and therefore the additional error is arbitrarily small.[3] We avoid complicated formalisms and instead slightly relax the approximation ratios in theorem statements by adding an arbitrarily small value $\delta > 0$ whenever the proofs use (1).

4 A Required Node Becomes a Steiner Node

The variant of the minimum Steiner tree reoptimization problem where a node is declared to be a Steiner node (STP_ϵ^{R-}) is defined as follows.

[3] Note that in contrast to the error from Lemma 1, we *cannot* control the error of the given solutions.

Input : An instance $(G, R, c, \textsc{Opt}_\epsilon^{\text{old}}, t)$ of STP_ϵ^{R-}
Output : A Steiner tree T
while $\deg_{\textsc{Opt}_\epsilon^{\text{old}}}(t) = 1$ **do** `// We assume that either` $\epsilon = 0$ `or` $\deg_{\textsc{Opt}_\epsilon^{\text{old}}}(t) > 1$
  Set $t' := \text{child}(t)$; `// The node adjacent to` t `in` $\textsc{Opt}_\epsilon^{\text{old}}$
  Remove t from $\textsc{Opt}_\epsilon^{\text{old}}$ and R, rename t' to t, and set $t \in R$;
  `// Now` $(G, R, c, \textsc{Opt}_\epsilon^{\text{old}}, t)$ `is the changed instance`
Transform $\textsc{Opt}_\epsilon^{\text{old}}$ to a k-restricted solution $\textsc{Opt}_{\epsilon,k}^{\text{old}}$ such that $\text{opt}_{\epsilon,k}^{\text{old}} \leq (1+\epsilon_k)\text{opt}_\epsilon^{\text{old}}$, where ϵ_k tends to 0 for large enough k;
Set $T_1 := \textsc{Opt}_\epsilon^{\text{old}}$; `// Note that` $\text{opt}_\epsilon^{\text{old}} \leq \text{opt}_{\epsilon,k}^{\text{old}}$
Let $C_1^t, C_2^t, \ldots$ be the full components of $\textsc{Opt}_{\epsilon,k}^{\text{old}}$ such that $t \in V(C_i^t)$ for all i and $c(C_i^t) \leq c(C_j^t)$ for $i < j$;
Set $F := \textsc{Opt}_{\epsilon,k}^{\text{old}} - C_1^t - C_2^t - C_3^t$; `// Ignore` C_3^t `if it does not exist`
Set $T_2 := \textsc{Connect}(F)$;
Set $T_3 := \textsc{Guess}(3k)$;
Set $T = T_i$ with $i = \min \arg_{j \in \{1,2,3\}}\{c(T_j)\}$.

Algorithm 3: DeclareSteiner

Given: A parameter $\epsilon > 0$, a Steiner tree instance (G, R, c), a solution $\textsc{Opt}_\epsilon^{\text{old}}$ to (G, R, c) such that $\text{opt}_\epsilon^{\text{old}} \leq (1+\epsilon)\text{opt}^{\text{old}}$, and a node $t \in R$.
Solution: A Steiner tree solution to $(G, R \setminus \{t\}, c)$.

An instance of STP_ϵ^{R-} is a tuple $(G, R, c, \textsc{Opt}_\epsilon^{\text{old}}, t)$. If $\epsilon = 0$, we skip the index and write STP^{R-}. Without loss of generality we assume that c is a metric: we may use the metric closure since the local modification does not change G or c.

The algorithm DeclareSteiner starts with reducing the instance to one where the changed required node has a degree of at least two, using a known technique. Afterwards it transforms the given solution to a k-restricted Steiner tree (note that the order of these two steps is important). The remaining algorithm outputs the best of three solutions that intuitively can be described as follows: we either keep the old solution; or we remove up to three full components incident to t to obtain a partial solution that we complete again using Connect; or we guess a partial solution that is at least as large as the $3k$ largest full-components of an optimal solution and complete these components to a solution using the best available approximation algorithm.

The following theorem indicates that in general we have to require $\epsilon = 0$ for instances of STP_ϵ^{R-} with $\deg(t) = 1$.

▶ **Theorem 3.** *For an arbitrary $\epsilon > 0$, let $\mathcal{A}$ be a polynomial time α-approximation algorithm for STP_ϵ^{R-}. Then there is a polynomial time α-approximation algorithm for the Steiner tree problem.*

Proof. Given a Steiner tree instance (G, R, c), let opt^{new} be the cost of an optimal solution. We construct a STP_ϵ^{R-} instance $(G', R', c', \textsc{Opt}_\epsilon^{\text{old}}, t)$ from (G, R, c). We first compute a minimum spanning tree $\tilde{T}$ of $G[R]$. Note that $G[R]$ is a complete graph since we assume c to be metric, and $c(\tilde{T}) \leq 2\text{opt}^{\text{new}}$, as shown by Takahashi and Matsuyama [30]; we assume w. l. o. g. that $\alpha < 2$. We obtain G' by combining G and a new node t as follows. We set $V(G') := V(G) \cup \{t\}$ and $E(G') = E(G) \cup \{t, t'\}$ for a node $t' \in R$. Then we obtain c' from c by setting $c'(t, t') = c(\tilde{T}) \cdot (1-\epsilon)/\epsilon$ and forming the metric closure. We set $R' = R \cup \{t\}$ and obtain a solution $\textsc{Opt}_\epsilon^{\text{old}}$ to (G', R', c') by adding $\{t, t'\}$ to $\tilde{T}$. Finally, we obtain the Steiner tree T by applying $\mathcal{A}$ to $(G', R', c', \textsc{Opt}_\epsilon^{\text{old}}, t)$.

Observe that T cannot contain an edge incident to t, since all of those edges are more expensive than $\tilde{T}$. Therefore T is a Steiner tree of (G, R, c). Conversely, all Steiner trees of (G, R, c) are feasible solutions to $(G', R', c', \textsc{Opt}_\epsilon^{\text{old}}, t)$. We conclude that T provides an α approximation, i. e., T is a feasible solution to (G, R, c) and $c(T) \leq \alpha \text{opt}^{\text{new}}$.

To finish the proof, we have to show that $\textsc{Opt}_\epsilon^{\text{old}}$ was a valid solution given to $\mathcal{A}$, i. e., its cost $\text{opt}_\epsilon^{\text{old}}$ is at most a factor $(1+\epsilon)$ larger than optimum. Clearly, $\textsc{Opt}_\epsilon^{\text{old}}$ is a Steiner tree of (G', R', c'). Let opt^{old} be the cost of an optimal Steiner trees for (G', R', c').

$$\frac{\text{opt}_\epsilon^{\text{old}}}{\text{opt}^{\text{old}}} = \frac{c(\tilde{T}) + c(t,t')}{\text{opt}^{\text{new}} + c(t,t')} \leq \frac{2\text{opt}^{\text{new}} + c(t,t')}{\text{opt}^{\text{new}} + c(t,t')} \leq 1 + \frac{\text{opt}^{\text{new}}}{\text{opt}^{\text{new}} + \text{opt}^{\text{new}} \cdot \frac{1-\epsilon}{\epsilon}} = 1 + \epsilon.$$

◀

For all remaining cases, DeclareSteiner profits from knowing $\textsc{Opt}_\epsilon^{\text{old}}$.

▶ **Theorem 4.** *Let $(G, R, c, \textsc{Opt}_\epsilon^{\text{old}}, t)$ be an instance of STP_ϵ^{R-} with $\deg_{\textsc{Opt}_\epsilon^{\text{old}}}(t) \geq 2$ or $\epsilon = 0$. Then, for an arbitrary $\delta > 0$, DeclareSteiner is an approximation algorithm for STP_ϵ^{R-} with performance guarantee*

$$\frac{(10\beta - 7 + 2\epsilon - 2\epsilon\beta)(1+\epsilon)}{7\beta - 4 + 5\epsilon - 2\epsilon\beta} + \delta.$$

For the approximation ratio $\beta = \ln(4) + \epsilon''$ from [24] with ϵ'' and δ chosen sufficiently small, we obtain an approximation ratio of less than $1.204 \cdot (1+\epsilon)$.

4.1 Proof of Theorem 4

Since k is a constant, all steps of DeclareSteiner except for the call of Connect clearly run in polynomial time. To see that also the call of Connect does, observe that removing the edges and Steiner nodes of a full component increases the number of connected components by at most $k-1$.

We continue with showing the claimed upper bound on the performance guarantee. Before we show the main result, we introduce two simplification steps. First, we show that we can restrict our attention to the case $\deg(t) = 2$ in $\textsc{Opt}_{\epsilon,k}^{\text{old}}$. Our analysis simultaneously gives a new proof for the previous best reoptimization result [13]. Subsequently we reduce the class of considered instances to those where all optimal solution to $(G, R \setminus \{t\}, c)$ have a special structure.

We start with analyzing the case where $\deg(t) = 1$. If this case appears in the while loop, by our assumption we have $\epsilon = 0$ and thus $\textsc{Opt}_\epsilon^{\text{old}}$ is an optimal solution. The transformation of DeclareSteiner within the while loop reduces the instance to one where $\deg(t) \geq 2$ [16]. When transforming the resulting solution $\textsc{Opt}_\epsilon^{\text{old}}$ to $\textsc{Opt}_{\epsilon,k}^{\text{old}}$, generally t could become a degree-one vertex. We use, however, that this is not the case when applying the algorithm of Borchers and Du [19]: The algorithm considers the full components separately, which implies that initially the degrees of all required vertices are one. Each full component is replaced by a graph where each required vertex has a degree of at least one. Consequently, the degree of no required vertex is decreased.

For the remaining proof, we assume $\deg_{\textsc{Opt}_{\epsilon,k}^{\text{old}}}(t) \geq 2$. We prove the following technical lemma, which is needed for our subsequent argumentation.

▶ **Lemma 5.** *There is a collection $\mathcal{C}$ of at most $3k$ full components of $\textsc{Opt}_k^{\text{new}}$ such that $F \cup \mathcal{C}$ is a connected graph.*

Proof. Observe that F has less than $3k$ connected components, and each of them contains nodes from R. We use that the full components of $\textsc{Opt}_k^{\text{new}}$ only intersect in R. Since $\textsc{Opt}_k^{\text{new}}$ is connected, by the pigeonhole principle it has a full component C that contains required nodes from two distinct components of F. Thus adding C to F reduces the number of components. Now the claim follows inductively. ◀

Let $\alpha = c(T)/\text{opt}^{\text{new}} \geq 1$ be the performance ratio of DeclareSteiner. Thus, in the following we want to determine an upper bound on α. We may assume

$$\text{opt}_{\epsilon,k}^{\text{old}} \geq \alpha \text{opt}^{\text{new}} \tag{2}$$

since otherwise, T_1 already gives an approximation ratio better than α.

We define $\gamma = c(\textsc{Connect}(F)) - c(F)$, the cost to connect F. Let d be the number of full components removed from $\textsc{Opt}_{\epsilon,k}^{\text{old}}$ to obtain F, i. e., $d \in \{2, 3\}$.

▶ **Lemma 6.** *For an arbitrary $\delta > 0$, the performance ratio α of* DeclareSteiner *is bounded from above by $1 + \frac{\beta-1+\epsilon(\beta-1)(d+1)}{1+(\beta-1)d+\epsilon} + \delta$.*

Proof. We have $c(C_1^t) + c(C_2^t) + c(C_3^t) \geq d \cdot c(C_1^t)$ assuming $c(C_1^t) \leq c(C_i^t)$ for $i \leq d$.

We determine the following constraints. Since $C_1^t + \textsc{Opt}^{\text{new}}$ contains a feasible solution to (G, R, c),

$$\text{opt}^{\text{new}} + c(C_1^t) \geq \text{opt}^{\text{old}}. \tag{3}$$

Furthermore,

$$\text{opt}_{\epsilon,k}^{\text{old}} - c(C_1^t) - c(C_2^t) - c(C_3^t) + \gamma \geq \alpha \text{opt}^{\text{new}} \tag{4}$$

since $c(T_2)$ is at most as large as the left hand side of (4).

We assume $\text{opt}_{\epsilon,k}^{\text{old}}$ tends to $\text{opt}_{\epsilon}^{\text{old}}$ for large enough k and then use (3) to replace opt^{old} in (2) to obtain

$$c(C_1^t) \geq \frac{\alpha - 1 - \epsilon}{1 + \epsilon} \text{opt}^{\text{new}}. \tag{5}$$

By applying (3) and (5) to (4), we obtain

$$\gamma \geq \frac{d}{1 + \epsilon} \cdot (\alpha - 1 - \epsilon) \text{opt}^{\text{new}}. \tag{6}$$

Finally, $\zeta \geq \gamma/\text{opt}_k^{\text{new}}$, by Lemma 5. Therefore, due to Lemma 2 and assumption that ϵ due to transformation to k-restricted tree tends to zero for large enough k,

$$\beta - \beta\gamma/\text{opt}_k^{\text{new}} + \gamma/\text{opt}_k^{\text{new}} \geq \alpha. \tag{7}$$

Now the claim follows if we assume $\text{opt}_k^{\text{new}}$ tends to opt^{new} for large enough k and replace γ in (7) by the right hand side of (6), where we used that $\beta \geq 1$. ◀

We note that for $d = 2$ and $\epsilon = 0$, the upper bound on the performance guarantee due to Lemma 6 matches the previously best performance guarantee [13]. For $d = 3$, the value is better than the aimed-for value from Theorem 4. Observe that a straightforward extension of DeclareSteiner would allow us to consider values of d larger than three.

Due to Lemma 6, in the following we may assume that $\deg(t) = 2$. Next, we analyze the structure of $\textsc{Opt}_{\epsilon,k}^{\text{old}}$ and $\textsc{Opt}^{\text{new}}$. Let $R_1 = (R \cap V(C_1^t)) \setminus \{t\}$ and $R_2 = (R \cap V(C_2^t)) \setminus \{t\}$.

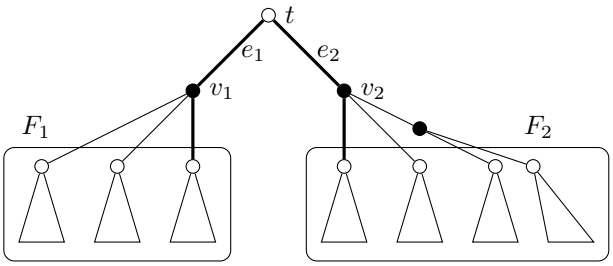


Figure 1 Structure of $\text{OPT}^{\text{old}}_{\epsilon,k}$. The paths P_1 and P_2 are drawn with thick lines.

We partition F into forests F_1 and F_2 such that F_1 contains exactly the trees T of F with $V(T) \cap R_1 \neq \emptyset$ and F_2 contains the remaining trees T', with $V(T') \cap R_2 \neq \emptyset$ (see Fig. 1).

Let $v_1 \in V(C_1^t)$ and $v_2 \in V(C_2^t)$ such that $e_1 = \{t, v_1\} \in E(C_1^t)$ and $e_2 = \{t, v_2\} \in E(C_2^t)$. Let P_1 be a minimum cost path in $\text{OPT}^{\text{old}}_{\epsilon,k}$ from t to R_1 and let P_2 be a minimum cost path in $\text{OPT}^{\text{old}}_{\epsilon,k}$ from t to R_2. Observe that P_1 contains e_1 and that P_2 contains e_2. We define $\kappa_1 := c(P_1)$, $\kappa_1' := c(e_1)$, and $\kappa_1'' = c(P_1) - c(e_1)$. Analogously, $\kappa_2 := c(P_2)$, $\kappa_2' := c(e_2)$, and $\kappa_2'' = c(P_2) - c(e_2)$. Note that we do not exclude that $v_1 \in R_1$ or $v_2 \in R_2$. In this case κ_1'' resp. κ_2'' are zero.

To simplify the presentation, we define $\kappa' := (\kappa_1' + \kappa_2')/2$ and $\kappa'' := (\kappa_1'' + \kappa_2'')/2$. Since P_1, P_2 are minimum cost paths, $c(C_1^t) \geq \kappa_1' + 2\kappa_1''$ and $c(C_2^t) \geq \kappa_2' + 2\kappa_2''$, which implies

$$c(C_1^t) + c(C_2^t) \geq 2\kappa' + 4\kappa''. \tag{8}$$

We have $\text{opt}^{\text{new}} + \kappa_1' + \kappa_1'' \geq \text{opt}^{\text{old}}$ and $\text{opt}^{\text{new}} + \kappa_2' + \kappa_2'' \geq \text{opt}^{\text{old}}$. Therefore,

$$\text{opt}^{\text{new}} + \kappa' + \kappa'' \geq \text{opt}^{\text{old}}. \tag{9}$$

▶ **Lemma 7.** *Suppose there are at least two edge disjoint paths in* $\text{OPT}^{\text{new}}_k$ *between* $V(F_1)$ *and* $V(F_2)$*. Then, for an arbitrary* $\delta > 0$*, the performance guarantee of* DeclareSteiner *is bounded from above by* $\frac{(11\beta-8)(1+\epsilon)}{8\beta-5+3\epsilon} + \delta$*.*

Proof. Let P' and P'' be two edge-disjoint paths within $\text{OPT}^{\text{new}}_k$ between $V(F_1)$ and $V(F_2)$ such that none of their internal nodes are in $V(\text{OPT}^{\text{old}}_{\epsilon,k})$. Without loss of generality, we assume that $c(P') \leq c(P'')$. We will also assume that $\text{opt}^{\text{new}}_k$ tends to opt^{new} for large enough k. Then, additionally to the previous constraints, we obtain the following.

$$\text{opt}^{\text{old}}_{\epsilon,k} - 2\kappa' + c(P') = \text{opt}^{\text{old}}_{\epsilon,k} - \kappa_1' - \kappa_2' + c(P') \geq \alpha \text{opt}^{\text{new}} \tag{10}$$

$$\zeta \cdot \text{opt}^{\text{new}}_k \geq c(P') + c(P'') \geq 2c(P') \tag{11}$$

From (9) and (10) and assuming $\text{opt}^{\text{old}}_{\epsilon,k}$ tends to $\text{opt}^{\text{old}}_{\epsilon}$ for large enough k, we obtain

$$c(P') \geq (\alpha - 1 - \epsilon)\text{opt}^{\text{new}} + (1-\epsilon)\kappa' - (1+\epsilon)\kappa'', \tag{12}$$

and thus, due to (11) and (2),(9),

$$\zeta \text{opt}^{\text{new}}_k \geq 2((\alpha-1-\epsilon)\text{opt}^{\text{new}} + (1-\epsilon)\kappa' - (1+\epsilon)\kappa'') \geq \frac{4}{(1+\epsilon)}(\alpha-1-\epsilon)\text{opt}^{\text{new}} - 4\kappa''. \tag{13}$$

Furthermore, by using (8) and (9) in (4), we obtain

$$\gamma \geq (\alpha - 1 - \epsilon)\text{opt}^{\text{new}} + (1-\epsilon)\kappa' + (3-\epsilon)\kappa''.$$

and thus, due to (2) and (9) and the fact that $\zeta \text{opt}_k^{\text{new}} \geq \gamma$,

$$\zeta \text{opt}_k^{\text{new}} \geq \frac{2}{(1+\epsilon)}(\alpha - 1 - \epsilon)\text{opt}^{\text{new}} + 2\kappa''. \tag{14}$$

A linear combination of (13) and (14) with coefficients one and two gives $\zeta \geq \frac{8(\alpha-1-\epsilon)}{3(1+\epsilon)}$, by assuming that $\text{opt}_k^{\text{new}}$ tends to opt^{new} for large enough k. Using (1) we obtain

$$\alpha \leq \frac{(11\beta - 8)(1+\epsilon)}{8\beta - 5 + 3\epsilon} + \delta. \qquad \blacktriangleleft$$

Since the value obtained by Lemma 7 is better than the aimed-for ratio, from now on we can restrict our focus to instances where in $\text{OPT}_k^{\text{new}}$, there are no two edge-disjoint paths between F_1 and F_2. In particular, this means that there is exactly one full component L in $\text{OPT}_k^{\text{new}}$ that connects F_1 and F_2. Since we assumed that there are no Steiner nodes of degree two in $\text{OPT}_k^{\text{new}}$, there is exactly one edge e_L in L such that removing e_L leaves two connected components of $\text{OPT}_k^{\text{new}}$, one containing R_1 and the other one containing R_2. Let P_L be a minimum cost path between $V(F_1)$ and $V(F_2)$ in L (and thus P_L clearly contains e_L). Let P_L^1 be the subpath of P_L between F_1 and e_L and let P_L^2 be the subpath of P_L between F_2 and e_L. We define $\lambda := c(P_L)$, $\lambda' := c(e_L)$, $\lambda_1'' := c(P_L^1)$, and $\lambda_2'' := c(P_L^2)$. Similar to above, we define $\lambda'' := (\lambda_1'' + \lambda_2'')/2$. Note that $\lambda - \lambda' = 2\lambda''$. It follows easily that $c(L) \geq \lambda' + 4\lambda''$. Let L' be a forest with a minimum number of full components from $\text{OPT}_k^{\text{new}}$ such that $\text{OPT}_{\epsilon,k}^{\text{old}} - C_1^t - C_2^t + L'$ is connected. From Lemma 5, we obtain that L' contains at most $3k$ full components and thus we considered guessing L' when computing T_3 in DeclareSteiner. We define $\xi := c(L') - \lambda' - 4\lambda''$. Since L' contains L, ξ is non-negative.

To find an upper bound on the value of α, we maximize α subject to the constraints (2), (9) and the following constraints.

By removing e_L from $\text{OPT}_k^{\text{new}}$ and adding the paths P_1 and P_2, we obtain a feasible solution to (G, R, c); conversely, by removing e_1 and e_2 from $\text{OPT}_{\epsilon,k}^{\text{old}}$ and adding P_L, we obtain a feasible solution to $(G, R \setminus t, c)$ that is considered in T_2. Therefore

$$\text{opt}_k^{\text{new}} + 2\kappa' + 2\kappa'' - \lambda' \geq \text{opt}^{\text{old}}, \tag{15}$$

$$\text{opt}_{\epsilon,k}^{\text{old}} - 2\kappa' + \lambda' + 2\lambda'' \geq \alpha \text{opt}^{\text{new}}. \tag{16}$$

In T_2 we also consider to remove C_1^t, C_2^t completely and to add L'. Therefore

$$\text{opt}_{\epsilon,k}^{\text{old}} - 2\kappa' - 4\kappa'' + \lambda' + 4\lambda'' + \xi \geq \alpha \text{opt}^{\text{new}}. \tag{17}$$

Due to Lemma 2 and assumption that ϵ due to transformation to k-restricted tree tends to zero for large enough k, we may assume

$$\beta - \beta\zeta + \zeta \geq \alpha. \tag{18}$$

In T_3, one of the considered guesses is L' and therefore

$$\zeta \cdot \text{opt}_k^{\text{new}} \geq \lambda' + 4\lambda'' + \xi. \tag{19}$$

We assume that $\text{opt}_k^{\text{new}}$ and $\text{opt}_{\epsilon,k}^{\text{old}}$ tends to opt^{new} and $\text{opt}_{\epsilon}^{\text{old}}$ respectively for large enough k and then scale the values such that $\text{opt}^{\text{new}} = 1$. Then we perform the following

replacements. We replace opt^{old} in (9) and in (15) by using (2); we use (9) to replace opt^{old} in (16); we use (9) to replace opt^{old} in (17). We keep (18) and (19). This way we obtain a linear program that maximizes α subject to the following constrains.

$$\begin{aligned} -\kappa' - \kappa'' + \alpha/(1+\epsilon) &\leq 1 \\ -2\kappa' - 2\kappa'' + \lambda' + \alpha/(1+\epsilon) &\leq 1 \\ (1-\epsilon)\kappa' - (1+\epsilon)\kappa'' - \lambda' - 2\lambda'' + \alpha &\leq 1+\epsilon \\ (1-\epsilon)\kappa' + (3-\epsilon)\kappa'' - \lambda' - 4\lambda'' - \xi + \alpha &\leq 1+\epsilon \\ (\beta-1)\zeta + \alpha &\leq \beta \\ \lambda' + 4\lambda'' + \xi - \zeta &\leq 0 \end{aligned}$$

Now we obtain the dual linear program

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 + (1+\epsilon)y_3 + (1+\epsilon)y_4 + \beta y_5 & \\ \text{s.t.:} \quad & -y_1 - 2y_2 + (1-\epsilon)y_3 + (1-\epsilon)y_4 & \geq 0 \\ & -y_1 - 2y_2 - (1+\epsilon)y_3 + (3-\epsilon)y_4 & \geq 0 \\ & y_2 - y_3 - y_4 + y_6 & \geq 0 \\ & -2y_3 - 4y_4 + 4y_6 & \geq 0 \\ & -y_4 + y_6 & \geq 0 \\ & (\beta-1)y_5 - y_6 & \geq 0 \\ & y_1/(1+\epsilon) + y_2/(1+\epsilon) + y_3 + y_4 + y_5 & \geq 1 \end{aligned}$$

To finish the proof, we consider the following feasible solution. We set

$$y_1 = \frac{2(\beta-1)(1+\epsilon)(1-2\epsilon)}{7\beta - 4 + 5\epsilon - 2\epsilon\beta}; \qquad y_2 = y_1/2;$$

$$y_3 = y_1/(1-2\epsilon); \qquad y_4 = y_1/(1-2\epsilon);$$

$$y_5 = \frac{3(1+\epsilon)(1-2\epsilon)}{(1-2\epsilon)(7\beta - 4 + 5\epsilon - 2\epsilon\beta)}; \qquad y_6 = \frac{3y_1}{2(1-2\epsilon)}.$$

With these values, the objective function value matches the claimed value in Theorem 4. By weak duality, we obtained an upper bound on the value of α in the primal linear program, which finishes the proof.

5 Increased Edge Cost

We now consider the reoptimization variant where the edge cost of one edge is increased, $\text{STP}_{\epsilon}^{E+}$. If e is the edge of G with increased cost, we define $c^{\text{new}}\colon E(G) \to \mathbb{R}_{\geq 0}$ as $c^{\text{new}}(e') = c(e')$ for all edges $e' \in E(G)\setminus\{e\}$ and $c^{\text{new}}(e)$ is the increased cost. Then the formal definition of the reoptimization variant is as follows.

Given: A parameter $\epsilon > 0$, a Steiner tree instance (G, R, c), a solution $\text{OPT}_{\epsilon}^{\text{old}}$ to (G, R, c) such that $\text{opt}_{\epsilon}^{\text{old}} \leq (1+\epsilon)\text{opt}^{\text{old}}$, and a cost $c^{\text{new}}(e) \geq c(e)$ for an edge $e \in E(G)$.
Solution: A Steiner tree solution to (G, R, c^{new}).

Observe that the cost function obtained after applying the local modification in general is not a metric, and $\text{OPT}_{\epsilon,k}^{\text{new}}$ is assumed to live in the metric closure according to the new cost function.

▶ **Theorem 8.** *Let* $(G, R, c, \text{OPT}_{\epsilon}^{\text{old}}, e, c^{\text{new}}(e))$ *be an instance of* $\text{STP}_{\epsilon}^{E+}$. *Then, for an arbitrary* $\delta > 0$, EdgeIncrease *is a* $\left(\frac{7\beta-4+\epsilon(4\beta-4)}{4\beta-1} + \delta\right)$*-approximation algorithm for* $\text{STP}_{\epsilon}^{E+}$.

Input : An instance $(G, R, c, \textsc{Opt}_\epsilon^{\text{old}}, e, c^{\text{new}}(e))$ of STP_ϵ^{E+}
Output : A Steiner tree T
Transform $\textsc{Opt}_\epsilon^{\text{old}}$ to a k-restricted solution $\textsc{Opt}_{\epsilon,k}^{\text{old}}$ such that $\text{opt}_{\epsilon,k}^{\text{old}} \leq (1+\epsilon_k)\text{opt}_\epsilon^{\text{old}}$ where ϵ_k tends to 0 for large enough k;
Set $T_1 := \textsc{Opt}_\epsilon^{\text{old}}$;
Set $T_2 := \textsc{Guess}(k+1)$, with respect to $c^{\text{new}}(e)$;
Set $T = T_i$ with $i = \min \arg_{j\in\{1,2\}}\{c(T_j)\}$.

Algorithm 4: EdgeIncrease

Proof. Let us introduce the following notation. To emphasize which of the two instances we consider, we write $c^{\text{old}}(e)$ instead of $c(e)$, where e is the edge with increased cost. We assume that $e \in E(\textsc{Opt}_{\epsilon,k}^{\text{old}})$, as otherwise T_1 would be good enough already. Therefore the graph $\textsc{Opt}_{\epsilon,k}^{\text{old}} - e$ has exactly two connected components F_1 and F_2. Similar to the previous proof, we define $R_1 := R \cap V(F_1)$ and $R_2 := R \cap V(F_2)$.

In $\textsc{Opt}_{\epsilon,k}^{\text{old}}$, let K be the full component that contains e. Let P be a minimum cost path from R_1 to R_2 within K. Then we set $\kappa := c(P) - c^{\text{old}}(e)$.

In $\textsc{Opt}_k^{\text{new}}$, there is a full component L of cost λ such that $V(L)$ contains nodes from both R_1 and R_2. If L has two edge-disjoint paths between R_1 and R_2, we define $\lambda' = 0$. Otherwise there is an edge $e_L \in E(L)$ such that e_L is a cut edge in $F_1 \cup F_2 \cup L$, and $\lambda' := c(e_L)$. We obtain the following inequalities, where as before $\alpha = c(T)/\text{opt}^{\text{new}}$.

Removing e and adding a shortest path between R_1 and R_2 within L gives a feasible solution to (G, R, c^{new}). Therefore T_2 is good enough unless

$$\text{opt}_{\epsilon,k}^{\text{old}} - c^{\text{old}}(e) + \lambda/2 + \lambda'/2 \geq \alpha\text{opt}^{\text{new}}. \tag{20}$$

One feasible solution to the original instance is to remove e_L and to add P. Therefore we obtain

$$\text{opt}_k^{\text{new}} - \lambda' + c^{\text{old}}(e) + \kappa \geq \text{opt}^{\text{old}}. \tag{21}$$

We obtain an additional constraint by observing that in addition to using e_L, within $\textsc{Opt}_k^{\text{new}}$ the required vertices of K have to be connected. Let K_1 be the tree of $K - e$ that contains $R_1 \cap V(K)$. We see K_1 as a rooted tree with the root r_1 contained in $e = \{r_1, r_2\}$. Let us fix any two vertices $u \neq u' \in V(K_1) \setminus \{r\}$, with parents v, v'. Then the minimum distance between the two subtrees rooted at u, u' is at least $\max\{c(u,v), c(u',v')\}$. The same argumentation holds for K_2, which we define analogous to K_1 (it contains $R_2 \cap V(K)$, and has the root r_2). By traversing a path from $V(K_1) \cap R_1$ to r_1 within K_1 and from $V(K_2) \cap R_2$ to r_2, and adding the distances, we conclude that there is a collection of at most k full components in $\textsc{Opt}_k^{\text{new}}$ that without counting e_L have a total cost of at least κ. Therefore, using T_2,

$$\zeta\text{opt}_k^{\text{new}} \geq \lambda' + \kappa.$$

From these constraints, we obtain the claimed result using arguments similar to those of the previous section. ◀

Acknowlegment. We would like to thank Anna Zych for some helpful comments and the anonymous reviewers for helping to improve the presentation.

References

1 Claudia Archetti, Luca Bertazzi, and Maria Grazia Speranza. Reoptimizing the traveling salesman problem. *Networks*, 42(3):154–159, 2003.

2 Claudia Archetti, Luca Bertazzi, and Maria Grazia Speranza. Reoptimizing the 0-1 knapsack problem. *Discrete Applied Mathematics*, 158(17):1879–1887, 2010.

3 Claudia Archetti, Gianfranco Guastaroba, and Maria Grazia Speranza. Reoptimizing the rural postman problem. *Computers & OR*, 40(5):1306–1313, 2013.

4 Giorgio Ausiello, Vincenzo Bonifaci, and Bruno Escoffier. *Complexity and approximation in reoptimization*. Imperial College Press/World Scientific, 2011.

5 Giorgio Ausiello, Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. *Journal of Discrete Algorithms*, 7(4):453–463, 2009.

6 Michael A. Bender, Martin Farach-Colton, Sándor P. Fekete, Jeremy T. Fineman, and Seth Gilbert. Reallocation problems in scheduling. In *Proc. of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2013)*, pages 271–279. ACM, 2013.

7 Tobias Berg and Harald Hempel. Reoptimization of traveling salesperson problems: Changing single edge-weights. In *Proc. of the Third International Conference on Language and Automata Theory and Applications (LATA 2009)*, volume 5457 of *LNCS*, pages 141–151. Springer, 2009.

8 Marshall W. Bern and Paul E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

9 Davide Bilò, Hans-Joachim Böckenhauer, Juraj Hromkovič, Richard Královič, Tobias Mömke, Peter Widmayer, and Anna Zych. Reoptimization of Steiner trees. In *Proc. of the 11th Scandinavian Workshop on Algorithm Theory (SWAT 2008)*, volume 5124 of *LNCS*, pages 258–269, 2008.

10 Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Królovic, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2):227–251, 2011.

11 Davide Bilò, Peter Widmayer, and Anna Zych. Reoptimization of weighted graph and covering problems. In *Proc. of the 6th International Workshop on Approximation and Online Algorithms (WAOA 2008)*, volume 5426 of *Lecture Notes in Computer Science*, pages 201–213. Springer, 2008.

12 Davide Bilò and Anna Zych. New reoptimization techniques employed to Steiner tree problem. In *Proceedings of the 6th Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS 11)*, 2011.

13 Davide Bilò and Anna Zych. New advances in reoptimizing the minimum Steiner tree problem. In *Mathematical Foundations of Computer Science 2012*, pages 184–197. Springer, 2012.

14 Hans-Joachim Böckenhauer, Luca Forlizzi, Juraj Hromkovič, Joachim Kneis, Joachim Kupke, Guido Proietti, and Peter Widmayer. On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research*, 2(2):83–93, 2007.

15 Hans-Joachim Böckenhauer, Karin Freiermuth, Juraj Hromkovič, Tobias Mömke, Andreas Sprock, and Björn Steffen. The Steiner tree reoptimization problem in graphs with sharpened triangle inequality. *Journal of Discrete Algorithms*, 11:73–86, 2012.

16 Hans-Joachim Böckenhauer, Juraj Hromkovič, Richard Královič, Tobias Mömke, and Peter Rossmanith. Reoptimization of Steiner trees: Changing the terminal set. *Theoretical Computer Science*, 410(36):3428–3435, 2009.

17 Hans-Joachim Böckenhauer, Juraj Hromkovič, Tobias Mömke, and Peter Widmayer. On the hardness of reoptimization. In *Proc. of the 34th International Conference on Current*

Trends in Theory and Practice of Computer Science (SOFSEM 2008), volume 4910 of *LNCS*, pages 50–65, 2008.

18 Hans-Joachim Böckenhauer and Dennis Komm. Reoptimization of the metric deadline TSP. *Journal of Discrete Algorithms*, 8(1):87–100, 2010.

19 Al Borchers and Ding-Zhu Du. The k-Steiner ratio in graphs. *SIAM Journal on Computing*, 26(3):857–869, 1997.

20 Nicolas Boria and Federico Della Croce. Reoptimization in machine scheduling. *Theoretical Computer Science*, 540:13–26, 2014.

21 Nicolas Boria, Jérôme Monnot, and Vangelis Th Paschos. Reoptimization of maximum weight induced hereditary subgraph problems. *Theoretical Computer Science*, 514:61–74, 2013.

22 Nicolas Boria and Vangelis T Paschos. A survey on combinatorial optimization in dynamic environments. *RAIRO-Operations Research*, 45(03):241–294, 2011.

23 Nicolas Boria and Vangelis Th Paschos. Fast reoptimization for the minimum spanning tree problem. *Journal of Discrete Algorithms*, 8(3):296–310, 2010.

24 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1):6, 2013.

25 S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971/72.

26 Bruno Escoffier, Martin Milanič, and Vangelis Th. Paschos. Simple and fast reoptimizations for the Steiner tree problem. *Algorithmic Operations Research*, 4(2):86–94, 2009.

27 Stefan Hougardy, Jannik Silvanus, and Jens Vygen. Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm. *arXiv preprint arXiv:1406.0492*, 2014.

28 Jérôme Monnot. A note on the traveling salesman reoptimization problem under vertex insertion. *Inf. Process. Lett.*, 115(3):435–438, 2015.

29 Markus W Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1):155–166, 1997.

30 Hiromitsu Takahashi and Akira Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24(6):573–577, 1979/80.

31 Anna Zych. *Reoptimization of NP-hard problems.* PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20257, 2012, 2012.

Minimizing Weighted ℓ_p-Norm of Flow-Time in the Rejection Model

Anamitra Roy Choudhury[1], Syamantak Das[2], and Amit Kumar[2]

1 **IBM Research, India**
anamchou@in.ibm.com
2 **IIT Delhi, India**
{sdas,amitk}@cse.iitd.ernet.in

Abstract

We consider the online scheduling problem to minimize the weighted ℓ_p-norm of flow-time of jobs. We study this problem under the *rejection model* introduced by Choudhury et al. (SODA 2015) – here the online algorithm is allowed to not serve an ε-fraction of the requests. We consider the restricted assignments setting where each job can go to a specified subset of machines. Our main result is an immediate dispatch non-migratory $1/\varepsilon^{O(1)}$-competitive algorithm for this problem when one is allowed to reject at most ε-fraction of the total weight of jobs arriving. This is in contrast with the speed augmentation model under which no online algorithm for this problem can achieve a competitive ratio independent of p.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, Flow time, Scheduling problem, Rejection model

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.25

1 Introduction

The problem of minimizing average flow-time, also known as response-time or waiting time, is of central importance in the scheduling literature [20, 5, 15, 2]. In the online setting of this problem, jobs arrive over time and need to be scheduled on machines, which may have varying characteristics. The flow-time of a job is defined as the difference between its completion time and release date, and we would like the jobs to have small flow-time. One way of measuring this is to take ℓ_p-norm of the flow-time of jobs, where the parameter p could vary depending on the particular application – varying p would mean balancing the trade-off between fairness and average response time. In this paper, we shall consider the well-studied subset-parallel (i.e., the restricted assignment) model, where each job j specifies a processing requirement p_j, but can be processed on a subset of the machines only.

The framework of competitive analysis for such problems turns out to be too pessimistic – it is known that there is no online algorithm for minimizing the average flow-time of jobs in the restricted assignment setting (even if we restrict all job sizes to 1) [16]. One popular approach toward handling this negative result is by providing the online algorithm slightly more power than the off-line adversary. Kalyanasundaram and Pruhs [19] introduced the speed-augmentation model where the machines of the online algorithm have slightly more speed than those of the offline algorithm. The speed augmentation model has been very successful in analyzing performance of natural algorithms for minimizing average flow-time in various scheduling settings. Anand et al. [4] showed that a natural greedy algorithm is constant competitive for minimizing average (weighted) flow-time in the restricted assignment setting

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 25–37

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(and in the more general unrelated machines setting) if we provide the online algorithm with $(1+\varepsilon)$-extra speed. Im and Moseley [18] extended this result to give an $O(p/\varepsilon^2)$-competitive algorithm for minimizing the ℓ_p-norm of weighted flow-time. However, they showed that the linear dependence on p was necessary (for any immediate dispatch algorithm) even if we allow constant-speedup and all weights are 1. In the extreme case when p becomes infinity, it is known that one cannot obtain better than $O(\log m)$-competitive algorithm (where m denotes the number of machines) even if we allow constant speed-up.

To address the apparent inability of the speed-augmentation model to handle large values of p, Choudhury et al. [13] considered a different job rejection model. Here, we allow the online algorithm to reject an ε-fraction of the jobs, where ε is an arbitrary small positive constant, whereas the off-line optimum is required to schedule all the jobs. This model seems to give more power to the online algorithm than that by the speed augmentation model – the latter model gives *uniformly* extra speed to all machines, whereas in the former model, we could trade-off across machines (by rejecting more jobs assigned to a particular machine at the expense of fewer rejected jobs to other machines). Choudhury et al. [13] formalized this intuition by giving a constant competitive algorithm for the problem of minimizing weighted ℓ_∞-norm of flow-time if the online algorithm is allowed to reject ε-fraction of the weight of the jobs. In this paper, we extend this result by showing that one can get a constant competitive algorithm even for the problem of minimizing weighted ℓ_p-norm of flow-time of jobs, if we are allowed to reject ε-fraction of the weight of the jobs. Note that the competitive ratio has no dependence on p, and so, we get a stronger result as compared to the speed-augmentation model (though we seem to provide more power to the online algorithm).

Our algorithm is based on reducing the problem of minimizing ℓ_p-norm to that of minimizing ℓ_∞-norm (with some more job rejections). But this requires one to track the *average* ℓ_p-norm of jobs released so far (in an off-line optimum algorithm). This turns out to be non-trivial, as this quantity could go up or down with time, and tracking it while not exceeding the optimal value at any time forms the heart of our algorithm. We state the main theorem of the paper as follows.

▶ **Theorem 1.** *If the online algorithm is allowed to reject ε-fraction of the weight of the jobs arrived so far, then there exists an $O(1/\varepsilon^{12})$-competitive algorithm for the problem of minimizing weighted ℓ_p-norm of the flow-time of jobs in the restricted assignments setting.*

Organization of the paper. In Section 2, we formally describe the problems considered in this paper. For sake of clarity, we give details of the special case when p is 1 (i.e., the average flow-time), and all weights are 1. The extension to the weighted case carries over using ideas in Choudhury et al. [13], and the result for ℓ_p-norm for arbitrary p follows without much changes – details of these changes are described in Section 7. In Section 4, we give an overview of the new ideas in this paper. The scheduling algorithm is described in Section 5. The algorithm is split in two parts – algorithm $\mathcal{A}$ first ensures that all queues are bounded, and subsequently, we give algorithm $\mathcal{B}$ which uses $\mathcal{A}$ to construct the actual schedule. Analysis of these algorithms is given in Section 6.

2 Problem Statement

We consider the online problem of scheduling jobs over multiple machines in the subset-parallel (i.e., the restricted assignment) setting. Here, jobs arrive over time. Each job j specifies a processing requirement p_j and a subset S_j of the machines on which it can be processed. Let r_j denote the release date (i.e., the arrival time) of job j. In the weighted

version of the problems, each job j also specifies a non-negative weight w_j. In this paper, we consider algorithms which follow the immediate-dispatch policy: when a job j arrives at time r_j, it is dispatched to one of the machines. Recall that we do not allow migration of jobs across machines, and so, the job gets processed on the machine to which it gets dispatched.

Given a schedule $\mathcal{S}$, the flow-time of a job in the schedule $\mathcal{S}$, $F^{\mathcal{S}}(j)$ is defined as the difference between its completion time in $\mathcal{S}$ and r_j. The goal of our algorithm is to minimize the weighted ℓ_p-norm of the flow-time of jobs, defined as $(\sum_j w_j F^{\mathcal{S}}(j)^p)^{1/p}$. We allow the online algorithm to reject an ε-fraction of the total weight of the jobs – note that the algorithm could reject a job immediately on its arrival, or much later after dispatching the job to a machine. Here ε is a positive (small enough) constant. Thus, we only consider the total flow-time of jobs which do not get rejected by the online algorithm. However, the optimal off-line algorithm is required to schedule all the jobs.

In this paper, we give details of the more special `SumFlowTime` problem, where we seek to minimize the total sum of the flow-time of jobs (i.e., all job weights are 1, and p is 1). The extension to the general case follows along predictable lines and is outlined in Section 7.

3 Related Work

There has been considerable work on scheduling with the objective of minimizing a suitable norm of the flow-time of jobs. For the objective of average flow-time of jobs, a logarithmic competitive algorithm in the identical machines setting is known [20, 5]. Garg and Kumar [15] and subsequently Anand et al. [2] extended this result to the related machines setting. Garg and Kumar [16] showed that the problem becomes considerably harder in the restricted assignment setting and no online algorithm with bounded competitive ratio is possible. Bansal and Pruhs [8] showed that the competitive ratio can be as high as $\Omega(n^c)$ for the problem of minimizing ℓ_p (for any $1 < p < \infty$) norm, where n is the number of jobs, even for a single machine. For minimizing the maximum flow-time in the identical machines model, Ambühl and Mastrolilli [1] gave a simple 2-competitive algorithm. However, Anand et al. [3] showed that the competitive ratio of any online algorithm for the restricted assignment setting is as high as $\Omega(m)$, where m is the number of machines.

The speed augmentation was first proposed by Kalyanasundaram and Pruhs [19] who used it to get an $O(1/\varepsilon)$-competitive algorithm for minimizing flow time on a single machine in the non clairvoyant setting. Bansal and Pruhs [8] proved that several natural scheduling algorithms are $O(1/\varepsilon)$-competitive algorithm for minimizing ℓ_p norm (for any $1 < p < \infty$) of flow-time of jobs in the single machine setting. Golovin et al. [17] extended this result to parallel machines setting. Chekuri et al. [12] showed that the immediate dispatch algorithm of Avrahami and Azar [5] is also $O(1/\varepsilon)$-competitive for all ℓ_p norms ($p \geq 1$).

In the general setting of unrelated machines with speed augmentation, Chadha et al. [10] gave an $O(1/\varepsilon^2)$-competitive algorithm for minimizing the sum of flow-time of jobs, which was improved and extended to the case of ℓ_p norm of flow-time by Im and Moseley [18] and Anand et al. [4]. Im and Moseley [18] present an $O(p/\varepsilon^{2+2/p})$ immediate dispatch and non-migratory algorithm for minimizing the ℓ_p norm of weighted flow-time in unrelated machine; they also show that any immediate dispatch non migratory online algorithm with speed $s > 1$ has competitive ratio $\Omega(p/s)$. Anand et al. [3] showed that for the problem of minimizing weighted ℓ_p norm of flow time of jobs, one cannot obtain competitive ratio better than $\Omega\left(\frac{p}{\varepsilon^{1-O(1/p)}}\right)$ even with non-immediate dispatch. The last two lower bounds hold even in the restricted assignment model.

For minimizing the maximum (unweighted) flow time on unrelated machines, Anand et al. [3] gave a $O(1/\varepsilon)$-competitive, $(1+\varepsilon)$-speed algorithm; however their algorithm is not an

immediate dispatch algorithm. In fact, Azar et al. [6] showed that any immediate dispatch algorithm with constant speedup is $\Omega(\log m)$-competitive in the restricted assignment setting. In the maximum weighted flow-time case, this lower bound holds even if we allow non-immediate dispatch [3].

Scheduling with Rejection. There has been considerable work on online scheduling with job rejections in the prize collecting setting where one incurs an extra cost for non-scheduled jobs (see e.g. [9, 14, 7, 11]).

4 Our Techniques

Here we outline the main ideas of our algorithm (which we call algorithm $\mathcal{A}$). The first idea is to start with the result of Choudhury et al. [13]. They consider the same setting as ours – jobs arriving online in the subset parallel setting. Given parameters ε and T, their schedule processes all but ε-fraction of the jobs. Assuming that there a schedule for which the ℓ_∞-norm of the flow-time of jobs is at most T, they give an online algorithm where the flow-time of all the non-rejected jobs is at most $O(T/\varepsilon^2)$.

For our problem, let us assume that we know the number of jobs n, and the optimal value $T_1^\star$ of the total flow-time of these jobs. From this it follows that at least $(1-\varepsilon)$-fraction of the jobs will have flow-time at most $T^\star = T_1^\star/(\varepsilon n)$. Conversely, if we can have a schedule which ensures that all but ε-fraction of jobs have flow-time at most $T^\star$, then the total flow-time of jobs which are not rejected is $O(T_1^\star/\varepsilon)$. Thus, we have converted the problem of minimizing the ℓ_1-norm to that of minimizing the ℓ_∞-norm of flow-time. So it seems natural to apply the result of Choudhury et al. mentioned above to our problem. However there are two main issues:

- The result of [13] assumes that the parameter T is such that there is a schedule for which the maximum flow-time of *all* jobs is at most T. For us, we have a parameter $T^\star$ such that there is a (off-line) schedule for which the maximum flow-time of all but ε-fraction of the jobs is at most $T^\star/\varepsilon$. We prove a generalization of the result of [13], where the online algorithm can reject 7ε-fraction of the jobs, whereas we compare it with an offline schedule for minimizing maximum flow-time of all but ε-fraction of the jobs. This requires going through the calculations of [13] and making some subtle changes to accommodate the changed settings.
- The more serious issue turns out to be the fact that we really do not know the values $T^\star$. In [13], as is usual in such problems, one starts with a small guess T of $T^\star$ – whenever the algorithm rejects more than ε-fraction of the jobs, they double the guess T and start afresh. This ensures that the T will never go beyond twice the optimal value $T^\star$. Here, we cannot adopt this strategy. Suppose the input consists of two *phases*: an initial phase with lot of jobs such that they have high objective value (both in terms of ℓ_1 and ℓ_∞ norms), and a second phase where jobs arrive over a much longer period of time, and so their flow-time are small. Our algorithm will need to increase the value of T during the first phase. However, it cannot work with a high value of T during the second phase – otherwise, it may allow the jobs in the second phase to last much longer than in an optimal solution. The problem arises from the fact that $T^\star$ is defined as the ratio of two parameters, both of which change (in fact, increase) with time, and so, if we are trying to keep track of $T^\star$, we will need to both increase and decrease the estimate T.

We now describe the details about how we handle the second problem above. As mentioned above, we maintain a variable T which is supposed to track the value $T^\star = T_1^\star/(\varepsilon n)$. We

start with a slightly weaker goal: we want to maintain a schedule such that at all times the queue size (i.e., the total remaining processing time of the jobs in the queue) on any machine remains bounded by $T/\varepsilon^{O(1)}$. We divide the execution of our algorithm into *phases* – during a phase P, we shall not change the value of estimate T (denoted by $\mathbf{est}(P)$). During a phase, we shall work with the algorithm of Choudhury et al. [13] (by supplying it the estimate $T = \mathbf{est}(P)$). The phase will be terminated by one of the following two events:

Case 1: We reject too many jobs in this phase (i.e., at least $O(\varepsilon)$ times the number of job arrivals): here, we show that if N jobs arrive during this phase, then the optimal value of the ℓ_1 norm of these jobs is $\Omega(NT\varepsilon^3)$. This lower bound allows us to pay for the flow-time of these jobs incurred by our algorithm (which will be $NT/\varepsilon^{O(1)}$). Further, we can end this phase, and start a new phase P' with $\mathbf{est}(P') = cT$, where $c = 2/\varepsilon$. We call such phases *good* phases. Whenever a good phase ends at a time t, we push its state on a stack $\mathbf{S}$. More formally, we push a new entry e on the stack $\mathbf{S}$, where e is the tuple $(T, Q(t), J)$. Here, $Q(t)$ denotes the jobs which are waiting to be processed on the queues of one of the machines, and J is a set of jobs for which we can argue that the optimal value is large. Note that we have *shelved* the jobs $Q(t)$ on the stack, and we will start with empty queues in the next phase.

Case 2: A lot of jobs arrive during this phase: this is the worrying case, because if N jobs arrive during this phase, then we may have paid $\Omega(NT)$ for their total flow-time, but the optimum value could be much less. The only way to pay for these jobs would be to charge to the lower bound obtained from previous good phases (stored in the stack $\mathbf{S}$). Whenever we charge to a phase in the top of the stack, we pop it so that it does not get charged again. Now, we would like to end this phase and start a new phase with a smaller value of the estimate T. We face several obstacles here:

- The first obstacle is that if t denotes the current time, then we would need to reduce the number of jobs in the queues of the machines. Our analysis requires that for any phase P, the queue sizes remain bounded by about $\mathbf{est}(P)/\varepsilon^{O(1)}$. Therefore, we are going to reduce the value of $\mathbf{est}(P)$, then we may need to reduce the queue sizes as well. This would mean rejecting jobs in the queues of the machines. Now this can be done provided we do not reject too many jobs. Assuming this is the case, we can start a new phase with a reduced estimate of T/c. However, note that in beginning of the new phase, we will still have non-empty job queues. Therefore, Case 1 (for the new phase) above needs to take these jobs into account as well.
- In the discussion above, when we are trying to reduce the queue sizes at time t, suppose we are not able to do so (because we would end up rejecting too many jobs). Here, we argue that even the optimal ℓ_1-norm of the flow-time of jobs in this phase will be $\Omega(NT\varepsilon^3)$. Thus, this phase again behaves like a good phase, and we save its state on $\mathbf{S}$. Further, we start a new phase P' with estimate T again.

5 The scheduling algorithm

We first describe the algorithm $\mathcal{A}$ and then extend it to the actual scheduling algorithm. We now give all the details of $\mathcal{A}$. We maintain a variable T during the algorithm. The variable T will change in powers of a constant c. For a phase P, we shall use $s(P)$ and $e(P)$ to denote its starting and ending time. We also have a stack $\mathbf{S}$ which is initially empty, and the variable $\mathbf{e}_{\mathtt{top}}$ will denote the entry at the top of the stack. Each entry e in the stack will be a tuple corresponding a phase P: $(\mathbf{est}(P), Q(e(P)), J)$, where $Q(t)$ denotes the jobs in

Algorithm DispatchJob(Job j):

If j is T-big
Reject the job
Else
Let j be of class k.
If for all $i \in S_j$, $\texttt{load}_{i,k}(r_j) + p_j \geq \alpha \cdot T$.
Reject the job
Else
Dispatch j to machine $i \in S_j$ for which $\texttt{load}_{i,k}(r_j)$ is minimum.

Figure 1 Algorithm for dispatching a job.

the queues of all the machines at time t, and J will be a set of jobs (for which we will argue that the optimal value is also large).

We first describe the job dispatch rule. Some definitions first. Let β denote a constant (which will be roughly $O(1/\varepsilon)$). We say that a job j is of class k if p_j lies in the interval $[\beta^k, \beta^{k+1})$. For a machine i, time t, and class k, let $Q_{i,k}(t)$ denote the jobs of class k waiting in the queue of machine i at time t; and define the $\texttt{load}_{i,k}(t)$ as the total remaining processing time of the jobs in $Q_{i,k}(t)$. The job dispatch rule is described in Figure 1. A job is said to be T-*big* if its size is at least $T \cdot (\varepsilon/2)$ and T-*small* otherwise. Thus, the algorithm just considers the queue sizes on each machine corresponding to the class to which j belongs. If all such queues are already full to their limit αT (where $\alpha = O(1/\varepsilon^3)$), we reject the job, else we dispatch it to the one with the smallest load on it.

We now describe the rule according to which jobs are processed on a machine. This is identical to that in [13], but we give it here (in Figure 2) for sake of completeness. It tries to balance two aspects: (i) process small jobs first, and (ii) process jobs from that class for which the corresponding queue is close to its allowable limit. Finally, we describe the algorithm $\mathcal{A}$ in Figure 3. Let P denote the current phase, and P^{prev} denote the previous phase. The algorithm distinguishes two cases: (i) P^{prev} was a good phase, i.e., $\mathbf{est}(P^{prev}) \leq \mathbf{est}(P) = T$, or (ii) P^{prev} was a bad phase, i.e., $\mathbf{est}(P^{prev}) > \mathbf{est}(P)$. If the former case happens, the phase P begins with empty queues of all machines, whereas in the latter case, it begins with non-empty queue sizes. The variable P' is meant to be P^{prev} in the latter case, whereas it is undefined (or empty) in the former. The variable $A(P)$ keeps track of the set of jobs arrived so far in P, whereas the variable $A'(P)$ keeps track of the set of jobs arrived in both P and P'. The variable $R(P)$ denotes the set of jobs which get rejected during the current phase. The variables $a(P)$, $a'(P)$ and $r(P)$ respectively denote the cardinality of the sets $A(P)$, $A'(P)$ and $R(P)$. Let $P^{\mathbf{top}}$ denote the phase corresponding to the entry in the top of the stack $\mathbf{S}$. In case $\mathbf{est}(P^{\mathbf{top}})$ is T or T/c, we define $Q^{\mathbf{top}}$ to be $Q(e(P^{\mathbf{top}}))$, i.e., the jobs which were shelved to the stack during this phase. Otherwise $Q^{\mathbf{top}}$ is set empty.

We now discuss the various steps in $\mathcal{A}$. We start with the estimate T to be 0, and P as the current phase. Recall that P' denotes the previous phase if the previous phase was a bad phase, else it is empty. When a job j arrives, we will increment the counters $a(P)$, $a'(P)$ which counts the number of jobs arrived so far in P and $P \cup P'$ respectively. Normally, we will just call **DispatchJob(j)**. However, this procedure will reject j if j happens to T-big. Now if very few (i.e., $1/\varepsilon$) jobs have arrived so far, then we do not want to reject any

Algorithm ProcessJob(i,t):

$k^\star := \mathbf{argmax}_k \frac{\texttt{load}_{i,k}(t)}{\beta^k}$.
Process the earliest released job from the queue $Q_{i,k^\star}(t)$
(use a fixed tie-breaking rule).

Figure 2 Algorithm for deciding which job gets processed at time t on a machine i.

job. Thus, if this case happens in the initial period of this phase (when not many jobs have arrived), we simply end this phase, and start a new phase with a much higher estimate – this phase will be a good phase because this job's processing time serves as a good lower bound. Note one subtlety – we will consider job j again in the next phase (because we haven't called **DispatchJob**(j) yet).

In the algorithm, we define two procedures – **EndGoodPhase** and **EndBadPhase**. The first one assumes that the current phase has ended as a good phase, while the latter one assumes otherwise. The procedure **EndGoodPhase** just ends the current phase by pushing a new entry on the stack, resetting the value of T, and initializing all queues to empty. The second procedure simply resets the value of T, but does not disturb the queues – the jobs in these queues carry over to the next phase.

Finally, we have a procedure **QueuedJobs**(v), where v is a parameter. This procedure finds the minimal collection of jobs which need to be removed from each of the queues $Q_i(k,t)$, where t denotes the current time, in the reverse order in which they were added to these queues, such that the total remaining processing time of jobs in each of the queues is at most v. It returns the set of such jobs.

As discussed before, algorithm $\mathcal{A}$ tries to maintain a schedule such that for all machines i, class k and time t, the queues $Q_{i,k}(t)$ remains bounded by $\mathbf{est}(P)/\varepsilon^{O(1)}$, where P denotes the phase containing time t. Note that $Q_{i,k}(t)$ only counts the jobs which arrive over this phase (and may be the jobs which were present initially in the queues of the machines if the previous phase was a bad phase). It does not count the jobs which have been shelved in the stack $\mathbf{S}$. We will however prove a stronger property: for any processing class k and an estimate T, let J_T^k be the set of jobs of class k which arrived during phases P for which $\mathbf{est}(P)$ was T; and let $J_T = \cup_k J_T^k$ denote the set of all such jobs. Then, at any time t and machine i, the total remaining processing time of jobs in J_T^k which were dispatched to machine i remains bounded by $T/\varepsilon^{O(1)}$. The fact, however, by itself is not sufficient to guarantee that we can finish any job of J_T within $T/\varepsilon^{O(1)}$ time. We now present our actual algorithm (algorithm $\mathcal{B}$) which ensures that the flow time of every job of J_T (with some additional rejections over algorithm $\mathcal{A}$) is bounded to at most $T/\varepsilon^{O(1)}$.

The scheduling algorithm $\mathcal{B}$. We here state our final scheduling algorithm. We will be using the result of Choudhury et al. [13] for the `GenWtdMaxFlowTime` problem. In the `GenWtdMaxFlowTime` problem, a job j has two weights associated with it, the rejection-weight $w_j^{(r)}$ and flow-time-weight $w_1^{(f)}$; the first one is used for counting the rejection weight of rejected jobs, while the second one is used in the weighted flow-time expression. The objective of the problem is to minimize the maximum over all jobs j of $w_j^{(f)} F_j$, where F_j denotes the flow-time of job j in a schedule; and we are allowed to reject jobs of rejection-weight at most ε times the total rejection-weight of all the jobs. The objective value is compared with the offline optimum which is not allowed to reject any job. In order to describe their

Algorithm $\mathcal{A}$:

Initialize:
$T \leftarrow 0$, $P' \leftarrow \emptyset$; $a(P), a'(P), r(P) \leftarrow 0$; $A(P), A'(P), R(P) \leftarrow \emptyset$.

Phase(P):

1. When a job j arrives at current time t
   (i) If j is T-big and $a(P) \leq 1/\varepsilon$,
   call **EndGoodPhase**$(T, Q(t), A(P) \cup \{j\}, \lceil p_j \rceil)$.
   (ii) Else
   – Update $A(P) \leftarrow A(P) \cup \{j\}, a(P) \leftarrow a(P) + 1$
   – Update $A'(P) \leftarrow A'(P) \cup \{j\}, a'(P) \leftarrow a'(P) + 1$
   – call **DispatchJob**(j)
   – if this job gets rejected,
   update $R(P) \leftarrow R(P) \cup \{j\}, r(P) \leftarrow r(P) + 1$
2. If $r(P) \geq 7\varepsilon \cdot a'(P)$
   (i) call **EndGoodPhase**$(T, Q(t), A'(P), cT)$.
3. If $a(P) \geq (|Q(s(P)|) + |Q^{\mathbf{top}}|)/\varepsilon$
   (i) Reject the jobs in $Q(s(P)) \cup Q^{\mathbf{top}}$.
   (ii) If $\mathbf{est}(P^{\mathbf{top}}) = T$ or T/c, pop the stack $\mathbf{S}$.
   (iii) Let $J \leftarrow$ **QueuedJobs**(T/c).
   (iv) If $|J| \leq 7\varepsilon a(P)$
   Reject all jobs in J and Call **EndBadPhase**(T/c).
   (v) Else Call **EndGoodPhase**$(T, Q(t), A(P), T)$.

EndGoodPhase(T_1, J_1, J_2, T_2)**:**

1. Push (T_1, J_1, J_2) on the stack $\mathbf{S}$.
2. Update $T \leftarrow$ smallest value of c^k above or equal to T_2, for integer k.
3. Initialize all queues to empty.
4. Start a new phase P with
   $A(P), A'(P), R(P) \leftarrow \emptyset, a(P), a'(P), r(P) \leftarrow 0$.

EndBadPhase(T_1)**:**

1. Update $T \leftarrow T_1$.
2. The queues on all machines remain unchanged.
3. Set P' to be the current phase P.
4. Start a new phase P with $A'(P) \leftarrow A(P')$, $A(P) \leftarrow \emptyset$.
   $a'(P) \leftarrow |A'(P)|, R(P) \leftarrow \emptyset, r(P), a(P) \leftarrow 0$.

Figure 3 Algorithm $\mathcal{A}$.

result, we also need to define *flow-time-weight class* and *rejection-weight-density class*. A job j with processing time p_j, rejection-weight $w_j^{(r)}$ and flow-time-weight $w_j^{(f)}$ is of flow-time-weight class k if $2^k \leq w_j^{(f)} < 2^{k+1}$. Similarly, it is of rejection-weight-density class k if $2^k \leq w_j^{(r)}/p_j < 2^{k+1}$. For a job j with remaining processing time p'_j at some time during a schedule, its remaining weighted processing time is simply $w_j^{(f)} \cdot p'_j$. Then we have

▶ **Theorem 2** ([13]). *Suppose there is an immediate dispatch schedule for an instance of the* `GenWtdMaxFlowTime` *problem with the following property: for every flow-time-weight class k and rejection-weight-density-class k', time t and machine i, the total remaining weighted processing times of such jobs waiting in the queue of machine i at time t is at most T, for a parameter T. Then, one can construct another immediate dispatch schedule which dispatches each job to the same machine as in the given schedule, and which may reject some jobs of rejection weight $O(\varepsilon)$ times the total rejection weight of all jobs, such that the weighted flow-time of every job is at most T/ε^4.*

We now present algorithm $\mathcal{B}$. This algorithm first maps our instance $\mathcal{I}$ to an instance $\mathcal{I}'$ of the `GenWtdMaxFlowTime` problem and then invokes the above theorem to get a schedule for $\mathcal{I}'$. We show that the corresponding schedule for $\mathcal{I}$ has the desired properties.

Our algorithm $\mathcal{B}$ will emulate algorithm $\mathcal{A}$ – when a job j arrives, it is dispatched according to $\mathcal{A}$: if $\mathcal{A}$ rejects this job, $\mathcal{B}$ also rejects it; and if $\mathcal{A}$ dispatches it to machine i, then $\mathcal{B}$ also dispatches this job to i. Further, if j does not get rejected, $\mathcal{B}$ adds the job to the instance $\mathcal{I}'$ with the same release date and processing requirement. If the current time (at which j is released) belongs to a phase P of schedule $\mathcal{A}$, then we set $w_j^{(f)}$ to $1/\mathbf{est}(P)$. We set $w_j^{(r)}$ to 1. Now we invoke the theorem above to build a schedule for $\mathcal{I}'$. This schedule may reject some more jobs, but dispatches jobs to the same machine as in $\mathcal{B}$. Thus, we can use the same schedule for $\mathcal{I}$ as well. This completes the description of our scheduling algorithm.

6 Analysis

We here give the analysis of the schedules $\mathcal{A}$ and $\mathcal{B}$.

Algorithm $\mathcal{A}$. We first show that algorithm $\mathcal{A}$ does not reject too many jobs.

▶ **Lemma 3.** *Algorithm $\mathcal{A}$ rejects $O(\varepsilon)$-fraction of the jobs.*

Proof. We argue that the total number of jobs rejected in a phase P is at most $O(\varepsilon)$ times the total number of jobs released during this phase and the previous phase. Summing over all phases, this will prove the desired result.

Algorithm $\mathcal{A}$ employs the following job rejections within a particular phase:

(a) Whenever a job arrives, the **DispatchJob** routine may reject the job - by Step 2 of the algorithm, the total number of such jobs is at most $7\varepsilon a'(P) + 1 \leq 8\varepsilon a'(P)$, because we know that $a'(P) \geq 1/\varepsilon$ (indeed, if $a'(P) < 1/\varepsilon$, then it is easy to check that **DispatchJob** will reject a job j only is it is T-big. But then we should have terminated this phase in Step 1(i) of the algorithm.)

(b) Rejection of jobs in Step 3: Note that steps 3(i) and 3(iv) can be executed at most once during a phase, because after these steps, we will end this phase. The conditions in Step 3 clearly state that the number of such job rejections is $O(\varepsilon a'(P))$. ◀

Next we state the main technical property of algorithm $\mathcal{A}$. This is where we show that for good phases, we get a lower bound on the optimal solution as well. Suppose the entry $(\mathbf{est}(P), Q, S)$ be pushed to the stack $\mathbf{S}$ at the end of a good phase P. Then the following lemma 4 gives a lower bound on the optimal flow-time of the jobs of S. Since $A(P) \subseteq S$, the lower bound also holds for the jobs arriving in the phase P. The proof of this lemma is deferred to the full version; the proof uses ideas from [13], but requires many new details as well.

▶ **Lemma 4.** *Suppose the entry* $(\mathbf{est}(P), Q, S)$ *be pushed to the stack* **S** *at the end of a good phase* P*. The total flow-time of any (off-line) algorithms on the set of jobs in* S *is at least* $\varepsilon^2 \cdot |S| \cdot \mathbf{est}(P)/c$*.*

Recall that for any class k, and parameter T, J_T^k denotes the set of jobs of class k which arrived during those phases P for which $\mathbf{est}(P)$ was T; and J_T denotes $\cup_k J_T^k$.

▶ **Lemma 5.** *Algorithm* $\mathcal{A}$ *ensures that at any time* t*, machine* i *and class* k*, the total remaining processing time of jobs of* J_T^k*, which have been dispatched to a machine* i*, at a time* t *is at most* $O(\alpha T)$*.*

Proof. We shall prove some invariant properties of the stack **S**. The lemma will follow from these properties. Note that an entry in the stack was pushed in Steps 1(i), or 2(i) or 3(v). For each such entry e, let T_e denote the estimate for the corresponding phase. For the sake of writing down the invariants, we define a related quantity, T'_e as follows: if e was pushed on the stack due to Steps 1(i) or 2(i), we define $T'_e = T_e$. Else we set $T'_e = T_e/c$. Consider a time t, and let P denote the current phase, and T be $\mathbf{est}(P)$. Let the entries in the stack (from top to bottom order) be $e_1, \ldots, e_k$. Then, the following property holds: $T > T'_{e_1} > \ldots > T'_{e_k}$.

We prove this by induction on t. For $t = 0$, there is no entry in the stack, and so this statement is true trivially. Now, suppose this is true for some time t during a phase P, and again, let T denote $\mathbf{est}(P)$. If this phase ends in Steps 1(i) or 2(i), then we push another entry e in the stack with $T_e = T'_e = T$. Further, the estimate for the next phase is strictly larger than T. Therefore, the condition continues to hold in the next phase.

Now, suppose the current phase P ends in Step 3(iv). If the stack top entry e satisfied $T'_e = T/c$, then T_e would be T or T/c. Hence, we would have popped such an entry in Step 3(ii). So when this phase ends, the top entry e in the stack would have $T'_e \leq T/c^2$. The next phase would have estimate equal to T/c. Therefore, the invariant continues to hold in the next phase as well.

Finally, suppose the current phase ends in Step 3(v). As argued above, we will pop out any entry with $T'_e = T/c$. Further, we push a new entry e with $T'_e = T/c$, and the estimate for the next phase is T. Thus the invariant holds in this case as well.

The statement of the lemma is clearly true for jobs released in a particular phase (by the properties of the **DispatchJob** algorithm. Now the above invariant implies that for any parameter T, the jobs from J_T which are still alive (i.e., waiting to be processed) can come from at most two phases. Thus, the lemma is true. ◀

Algorithm $\mathcal{B}$. We now discuss the properties of algorithm $\mathcal{B}$. We first show that the flow-time of the jobs in a particular phase is bounded by the estimate for that phase.

▶ **Lemma 6.** *Algorithm* $\mathcal{B}$ *ensures that the flow time of every job of* J_T *(which is not rejected) is at most* $O(\alpha T/\varepsilon^4)$*.*

Proof. We use the notation while discussing algorithm $\mathcal{B}$. Since the rejection weights are all 1, the rejection-weight-density class essentially becomes the same as the definition of class based on processing time only. Assuming c is a power of 2, it follows that each flow-time-weight-class corresponds to a particular value of the estimate T (since the estimates are powers of c). Now, Lemma 5 shows that for a particular class k and estimate T, and a time t and machine i, the total remaining processing time of jobs from J_T^k at time t on machine i is at most $O(\alpha T)$. Hence, their total remaining weighted-processing time is $O(\alpha)$ – note that this bound holds for all jobs (irrespective of k and T). Applying Theorem 2, we get that the

schedule $\mathcal{B}$ incurs weighted flow-time of $O(\alpha/\varepsilon^4)$ for such jobs, and so, the actual flow-time is $O(\alpha T/\varepsilon^4)$. ◀

Having bounded the flow-time of jobs in our schedule, we now give the lower bound of the corresponding quantity for the off-line optimum. Our main technical lemma 4 already lower bounds the optimum value for a specific good phase. The following claim follows easily from this result.

▶ **Claim 7.** *The total flow-time of jobs released during good phases is at most $O(1/\varepsilon^{11})$ times the optimal value.*

Proof. The proof directly follows from Lemma 4 and Lemma 6, and the fact that $|A'(P)| \leq |S| \leq |A'(P)|$, where the entry $(\mathbf{est}(P), Q, S)$ is pushed to the stack $\mathbf{S}$ at the end of the good phase P, and $|A'(P)|$ is at most the number of jobs released during P and the previous phase. ◀

It remains to bound the flow-time of jobs released during bad phases. Before this, we make an important observation.

▶ **Claim 8.** *For any phase P, the set of jobs in $Q(s(P))$, i.e., the jobs waiting in the queues of the machines at the beginning of this phase, could have only arrived during the previous phase.*

Proof. Let P' be the phase preceding to P. If P' is a good phase, then P will start with empty queues, so there is nothing to prove. If P' is a bad phase, it will remove the jobs in $Q(s(P'))$ (in Step 3(i)), and so, the only jobs which carry over to phase P must have been released during P'. ◀

▶ **Lemma 9.** *The total flow-time of jobs released during bad phases is at most $O(1/\varepsilon^{12})$-times the optimal value.*

Proof. Let $B_1, \ldots, B_l$ be a maximal sequence of bad phases, and let G_0 denote the good phase preceding B_1. In Step 3 of the algorithm $\mathcal{A}$, each of the phases B_i may pop an entry from the stack – let this entry correspond to a good phase G_i. Note that G_0 is same as G_1. We know that $\mathbf{est}(G_i) \geq \mathbf{est}(B_i)/c$. For phase P, let $N(P)$ denote the number of jobs released during that phase. For phase B_i, we must have ended it when the condition in Step 3 was satisfied. Therefore, $N(B_i)$ is at most $(N(G_i) + N(B_{i-1}))/\varepsilon$, if $i \geq 1$ (using the above claim). Lemma 6 shows that the total flow-time of jobs during $B_1, \ldots, B_l$ is at most $\sum_{i=1}^{l} N(B_i) \cdot \frac{\alpha \mathbf{est}(B_i)}{\varepsilon^4}$. If T denotes $\mathbf{est}(B_1)$, then $\mathbf{est}(B_i) = \frac{T}{c^{i-1}}$. Therefore, the total flow-time of the jobs released during $B_1, \ldots, B_l$ is at most

$$\sum_{i=1}^{l} N(B_i) \cdot \frac{\alpha T}{c^{i-1}\varepsilon^4}. \tag{1}$$

We also know that $N(B_i) \leq 1/\varepsilon \cdot (N(G_i) + N(B_{i-1}))$. Since $c = 2/\varepsilon$, we get

$$N(B_i) \cdot \frac{\alpha T}{c^{i-1}\varepsilon^4} - N(B_{i-1}) \cdot \frac{\alpha T}{2c^{i-2}\varepsilon^4} \leq N(G_i) \cdot \frac{\alpha T}{2c^{i-2}\varepsilon^4}.$$

Summing the above for all i, we get

$$\sum_i N(B_i) \cdot \frac{\alpha T}{c^{i-1}\varepsilon^4} \leq 2c \sum_i N(G_i) \cdot \frac{\alpha T}{c^{i-1}\varepsilon^4}.$$

Now observe that $\mathbf{est}(G_i) \geq \frac{T}{c^i}$, and Lemma 4 implies that the total flow-time of the jobs released during $G_1, \ldots, G_l$ is at least $\sum_i \Omega(\varepsilon^3 \cdot N(G_i) \cdot \mathbf{est}(G_i))$, which is at least $\Omega(\varepsilon^{12})$ times the total flow-time of the jobs released during $B_1, \ldots, B_l$. Now, we sum this up over all maximal sequences of bad phases, and observe that G_i is uniquely determined by B_i(a stack entry once popped never gets pushed back again). ◀

7 Extension to weighted ℓ_p norm

We first outline the steps needed to extend our results to the case where each job j has a weight w_j. Let W denote the total weight of arriving jobs, and let $T_1^\star$ denote the optimal value $T_1^\star$ of the total weighted flow-time of these jobs. It is easy to see that jobs of total weight at least $(1-\varepsilon)W$ will have (unweighted) flow-time at most $T^\star = T_1^\star/(\varepsilon W)$. Thus, we modify algorithm $\mathcal{A}$ to maintain a variable T which is supposed to track the value $T^\star = T_1^\star/(\varepsilon W)$.

We say a job j with processing time p_j, and weight w_j is of *density class* k if $\beta^k \leq \frac{w_j}{p_j} < \beta^{k+1}$, for some constant β (which will be roughly $O(1/\varepsilon)$). The job dispatch and job processing rule is same as the algorithm for the unweighted case, with the only difference that now $Q_{i,k}(t)$ and $\mathtt{load}_{i,k}(t)$ are defined for every density class k. Rest of the details of $\mathcal{A}$ remain unchanged, except for the fact that $a(P), a'(P), r(P)$ now keep track of the total weight of corresponding jobs. The algorithm $\mathcal{B}$ remains unchanged except for the fact that for a job j, it sets $w_j^{(r)}$ to its weight w_j.

We now show how our results extend to the problem of minimizing the ℓ_p norm of the flow time of the jobs, for some positive constant p. For sake of clarity, we argue about the unweighted case only, though the weighted case follows similarly. Let us assume that we know the total number of the jobs released n, and the optimal value $T_1^\star$ of the ℓ_p norm of the flow-time of these jobs. From this it follows that at least $(1-\varepsilon)$-fraction of the total number of jobs will have (unweighted) flow-time at most $T^\star = T_1^\star/(\varepsilon n)^{1/p}$. Thus in algorithm $\mathcal{A}$, we maintain a variable T which is supposed to track the value $T^\star = T_1^\star/(\varepsilon n)^{1/p}$. Also, we increase or decrease T in factors of $c = (2/\varepsilon)^{1/p}$. The rest of the algorithms $\mathcal{A}$ and $\mathcal{B}$ remains as it is for the problem of minimizing the ℓ_1 norm of the flow time of jobs.

References

1 Christoph Ambühl and Monaldo Mastrolilli. On-line scheduling to minimize max flow time: an optimal preemptive algorithm. *Oper. Res. Lett.*, 33(6):597–602, 2005.

2 S. Anand. *Algorithms for flow time scheduling.* PhD thesis, Indian Institute of Technology, Delhi, December 2013.

3 S. Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. Minimizing maximum (weighted) flow-time on related and unrelated machines. In *ICALP (1)*, pages 13–24, 2013.

4 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.

5 Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, 2003.

6 Yossi Azar, Joseph (Seffi) Naor, and Raphael Rom. The competitiveness of on-line assignments. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 203–210, 1992.

7 Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Kedar Dhamdhere. Scheduling for flow-time with admission control. In *Proc. ESA, 2003*, 2003.

8 Nikhil Bansal and Kirk Pruhs. Server scheduling in the l_p norm: a rising tide lifts all boat. In *STOC*, pages 242–250, 2003.

9 Yair Bartal, Stefano Leonardi, Alberto Marchetti-Spaccamela, Jiri Sgall, and Leen Stougie. Multiprocessor scheduling with rejection. *SIAM J. Discrete Math.*, 13(1):64–78, 2000.

10 Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *STOC*, pages 679–684, 2009.

11 Ho-Leung Chan, Sze-Hang Chan, Tak Wah Lam, Lap-Kei Lee, and Jianqiao Zhu. Non-clairvoyant weighted flow time scheduling with rejection penalty. In *SPAA*, pages 246–254, 2012.

12 Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *STOC*, pages 363–372, 2004.

13 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. In *SODA*, pages 1114–1133, 2015.

14 Leah Epstein and Hanan Zebedat-Haider. Preemptive online scheduling with rejection of unit jobs on two uniformly related machines. *J. Scheduling*, 17(1):87–93, 2014.

15 Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP (1)*, pages 181–190, 2006.

16 Naveen Garg and Amit Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS*, pages 603–613, 2007.

17 Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. All-norms and all-ℓ_p-norms approximation algorithms. In *FSTTCS*, pages 199–210, 2008.

18 Sungjin Im and Benjamin Moseley. Online scalable algorithm for minimizing ;k-norms of weighted flow time on unrelated machines. In *SODA*, pages 95–108, 2011.

19 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. In *FOCS*, pages 214–221, 1995.

20 Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

On Correcting Inputs: Inverse Optimization for Online Structured Prediction*

Hal Daumé III, Samir Khuller, Manish Purohit, and Gregory Sanders

Computer Science Department
University of Maryland, College Park, MD, US
{hal,samir,manishp,gsanders}@cs.umd.edu

Abstract

Algorithm designers typically assume that the input data is correct, and then proceed to find "optimal" or "sub-optimal" solutions using this input data. However this assumption of correct data does not always hold in practice, especially in the context of online learning systems where the objective is to learn appropriate feature weights given some training samples. Such scenarios necessitate the study of inverse optimization problems where one is given an input instance as well as a desired output and the task is to adjust the input data so that the given output is indeed optimal. Motivated by learning structured prediction models, in this paper we consider inverse optimization with a margin, i.e., we require the given output to be better than all other feasible outputs by a desired margin. We consider such inverse optimization problems for maximum weight matroid basis, matroid intersection, perfect matchings in bipartite graphs, minimum cost maximum flows, and shortest paths and derive the first known results for such problems with a non-zero margin. The effectiveness of these algorithmic approaches to online learning for structured prediction is also discussed.

1998 ACM Subject Classification I.2.6 Learning

Keywords and phrases Inverse Optimization, Structured Prediction, Online Learning

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.38

1 Introduction

Algorithm designers generally assume that the input data is sacrosanct and correct. Algorithms are then typically run on this input data to compute "optimal" or "sub-optimal" solutions quickly whether it be the computation of a maximum spanning tree, a maximum matching, max weight arborescence, or shortest paths. However, with an increasing reliance on automatic methods to collect data, as well as in systems that learn, this assumption does not always hold. The input data can be erroneous (even though it may be approximately correct), and it becomes important to "adjust" the input data to achieve certain desired conditions.

A simple example can be used to illustrate the main point – suppose we are given a weighted graph $G = (V, E)$ and a spanning tree T, and told that T *should* be a maximum weight spanning tree in G. The goal now is to perturb the edge weights of the graph G, minimizing the L_2 norm of the perturbation, so that T is indeed the optimal spanning tree. This kind of problem has been studied previously in the form of "Inverse Optimization"

* Partially supported by NSF grants IIS-1451430 and CCF-1217890.

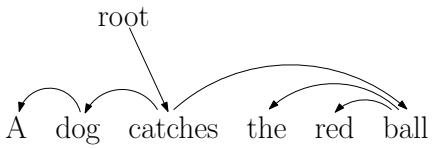


Figure 1 Example dependency parse tree. The tree describes the relations between head words and their dependents in the sentence.

problems. However, we wish to accomplish a stronger goal of making sure that the given tree T is better than *every* other tree in G by a given margin δ.

Our initial motivation for studying this problem comes from the *structured prediction* task in machine learning [15, 20, 3, 22, 24]. For concreteness and ease of exposition, we now describe structured prediction in the context of predicting dependency parse trees for natural language sentences. Given an English sentence, its dependency parse is a rooted, directed tree that indicates the dependencies between different words in the sentence as shown in Figure 1. The input sentence can be represented as a complete, directed graph on the words of the sentence that is parameterized by *features* on the edges. Given a learned model (represented as a vector of parameters), the weight of an edge is computed as the inner product of its feature vector and the model. As linguistic constraints dictate that the required dependency parse must form a rooted, spanning arborescence of the graph, one can use off-the-shelf combinatorial algorithms [9, 2] to find the highest weight arborescence. The learning problem is thus to find a parameter vector such that once the edges are weighted by the inner products, running a combinatorial optimization algorithm would return the desired parse tree. At "training time", we are given a sentence as well as its correct parse tree and the problem that we need to solve is exactly the inverse optimization problem - given the current model and the parse tree, say T, find the minimum perturbation to the model so that the combinatorial optimization algorithm would return T. It is well established in the learning theory literature that achieving a large margin solution enables better generalization [6]. We consider minimizing the L_2 norm because of connections to prior work [14]. In particular, for applications in structured prediction, the convergence and error bounds (included in Section 6) require L_2 norm minimization.

In our work we consider such inverse optimization problems with a margin in a general matroid setting. We consider both the problem of modifying the weights of the elements of a matroid, so that a given basis is a maximum weight basis (with a given margin of δ) and the considerably harder problem of matroid intersection where a given basis of two matroids should have weight higher (by at least δ) than any other set of elements that is a basis in the two matroids. This framework captures two special cases which are useful for structured prediction - namely maximum weight bipartite matching (useful for language translation) and maximum weight arborescence (useful for sentence parsing). We also consider δ-margin inverse optimization problems for a number of other classical combinatorial optimization problems such as perfect matchings, minimum cost flows and shortest path trees. In addition, we present a generic framework for online learning for structured prediction using the corresponding inverse optimization problem as a subroutine and present convergence and error bounds on this framework.

1.1 Related Work

Inverse optimization problems have been widely studied in the Operations Research literature. Most prior work however has focused on minimizing the L_1 or L_∞ norms between the weight

vectors and, more importantly, do not allow non-zero margin (δ). Heuberger [13] provides an excellent survey of the diverse inverse optimization problems that have been tackled. Both the inverse matroid optimization [8] and matroid intersection [16] have previously been studied in the setting of minimizing the L_1 norm and with zero margin. However, they use techniques that are specialized to minimizing the L_1 norm of the perturbation and do not extend to minimizing the L_2 norm. At the same time, these approaches to do not generalize to the general case of inverse optimization with non-zero margins.

In typical global models for structured prediction (for e.g. see [15, 17, 24, 3, 5, 18]), the discrete optimization problem is considered a "black box". By treating the combinatorial problem as a black box, these methods lose the ability to precisely reason about how certain changes to the underlying parameter vector can affect the eventual output. The simplest approach to solving the online structured prediction problem is the structured perceptron [3]. On each example, the structured perceptron makes a prediction based on its current model. If this prediction is incorrect, the algorithm suffers unit loss and updates its parameters with a simple linear update that moves the predictor closer to the truth and further from the current best guess. While empirically successful in a number of problems, this particular update is relatively imprecise: there are typically an exponential number of possible outputs for any given input, and simply promoting the correct one and demoting the models' current prediction may do very little to move the model as far as it needs to go. An alternative approach is the large margin discriminative approach [6] that seeks to change the parameters as little as possible subject to the constraint that the true output has a higher score than all incorrect outputs. However, such an approach is often computationally infeasible for structured prediction as there are usually an exponential number of potential outputs. McDonald et al. [18] circumvent this infeasibility by using a k-best list of possible outputs and restrict the set of constraints to require that the true output has a higher score than the incorrect outputs on the k-best list. This has been shown to be effective for small values of k on simple parsing tasks [18]. However, for more complex tasks, like machine translation, one needs more complicated update frameworks [1]. In this work we show that the large margin discriminative approach is applicable to a wide range of problems in structured prediction using techniques from inverse combinatorial optimization.

In the context of online prediction, the most related work to ours is that of Taskar et al. [22], who also consider structured prediction using inverse bipartite matchings. They define a loss function that measures, against a ground truth matching, the number of mispredicted edges in the found matching. This "Hamming distance" style loss function nicely decomposes over the structure of the graph and thereby admits an efficient "loss augmented" inference solution, in which correct edges are penalized during learning. (The idea is that if correct edges are penalized, but the model still produces the correct matching, then it has done so with a sufficiently large margin.) This idea only works in the case of decomposable loss functions, or the simpler 0-margin formulation. In comparison, our approach works both for decomposable loss functions as well as "zero/one loss" over the entire structure. Furthermore, our approach generalizes to arbitrary matroid intersection problems and minimum cost flows and thus is applicable to a much wider range of structured prediction problems.

1.2 Contribution and Techniques

A lot of prior work in the inverse optimization literature formulates the problem as a linear program and then uses strong duality conditions to find the new perturbed weights. However, such techniques cannot be extended to handle a non-zero margin that is required by the application to structured prediction. We formulate inverse optimization to minimize the

L_2 norm of the perturbations as a quadratic program and use problem specific optimality conditions to determine a concise set of linear constraints that are both necessary and sufficient to guarantee the required margin. In particular, one of the key ingredients is a set of polynomially many linear constraints that ensure that an appropriately defined auxiliary graph does not contain small directed cycles. We note that our formulations can easily be adapted to minimize the L_1 norm of the perturbations by simply modifying the objective and using linear programming.

We obtain concise formulations for exactly solving δ-margin inverse optimization problems for (i) maximum weight matroid basis, (ii) maximum weight basis in the intersection of two matroids, (iii) shortest s-t path, (iv) shortest path tree, (v) minimum cost maximum flow in a directed graph.

We also present convergence results for the generic online learning framework for structured prediction motivating our study.

The rest of the paper is organized as follows. In Section 2, we formally define δ-margin inverse optimization. In Sections 3 and 4, we present our results on inverse optimization for matroids, and matroid intersections respectively. In Section 5, we present a more efficient algorithm for the special case of inverse perfect matchings in bipartite graphs. In Section 6, we describe an online learning framework for structured prediction as an application. The proofs of convergence and error bounds for this learning framework as well as some preliminary experimental results for our learning model are included in the full version of this paper [7]. We also defer the results for inverse optimization for shortest path trees and minimum cost flow problems to the full version.

2 Problem Description

As explained in the introduction, we require a given solution to be better than all other feasible solutions by a margin of δ. We now formalize this notion of δ-optimality.

▶ **Definition 1** (δ-Optimality)**.** For a maximization problem P, let $\mathcal{F}$ denote the set of feasible solutions, let w be the weight vector, $c(w, A)$ denote the cost of feasible solution A under weights w, and let $\delta \geq 0$ be a scalar. A feasible solution $S \in \mathcal{F}$ is called δ-optimal under weights w if and only if

$$c(w, S) \geq c(w, S') + \delta, \quad \forall S' (\neq S) \in \mathcal{F}.$$

δ-optimality for minimization problems is defined similarly. All problems we consider in this work can be classified as δ-margin inverse optimization.

▶ **Definition 2** (δ-Margin Inverse Optimization)**.** For a given optimization problem P, let $\mathcal{F}$ denote the set of feasible solutions, let w be the weight vector, let $\delta \geq 0$ be a scalar, and let $S \in \mathcal{F}$ be a given feasible solution. δ-Margin Inverse optimization is to find a new weight vector w' minimizing $||w' - w||_2$ (L_2 norm) such that S is the δ-optimal solution of P under weights w'.

In the following sections we consider δ-margin inverse optimization for a number of problems mentioned earlier.

3 Maximum weight matroid basis

In order to provide intuition about the type of problems we propose to solve in this paper, we first begin with the simple case of Inverse Matroid Optimization. We recall the definition of a matroid.

▶ **Definition 3** (Matroid). A matroid is a pair $\mathbb{M} = (X, \mathcal{I})$ where X is a ground set of elements and $\mathcal{I}$ is a family of subsets of X (called Independent sets) such that

(i) $\mathcal{I} \neq \phi$.

(ii) (Hereditary) If $B \in \mathcal{I}$, and $A \subseteq B$, then $A \in \mathcal{I}$.

(iii) (Exchange property) If $A, B \in \mathcal{I}$, and $|A| < |B|$, then there exists some element $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

▶ **Definition 4** (Matroid Basis and Circuit). Let $\mathbb{M} = (X, \mathcal{I})$ be a matroid. Then any maximal independent set in $\mathcal{I}$ is called a basis of the matroid. Conversely, any minimal dependent set is called a circuit.

For the inverse problem we are given a matroid $\mathbb{M} = (X, \mathcal{I})$, a weight function w on the elements, and a basis B of $\mathbb{M}$. The goal is to find a weight function w' so that B is the δ-optimal basis of $\mathbb{M}$ under the new weights. As it is well known that a spanning tree is a basis of a graphical matroid, this inverse matroid optimization problem directly generalizes the inverse maximum spanning tree problem.

We first state a simple optimality condition for a given basis B of a matroid $\mathbb{M}$. An easy generalization of [21] for $\delta \geq 0$ gives the following lemma.

▶ **Lemma 5** (Corollary 39.12b in [21]). *A given basis B of a matroid $\mathbb{M}$ is δ-optimal (under weight function w) if and only if for any $f \notin B$, and each $e \in C_B(f)$, $w(e) - w(f) \geq \delta$, where $C_B(f)$ denotes the unique circuit in $B \cup \{f\}$.*

We thus have a set of polynomially many linear constraints that are necessary and sufficient for the given basis B to be δ-optimal. The inverse matroid optimization problem can then be formulated as a linearly constrained quadratic problem as follows -

$$\min_{w'} \sum_{e \in X} (w'(e) - w(e))^2 \quad \textbf{subj. to:} \tag{1}$$

$$w'(e) - w'(f) \geq \delta, \quad \forall f \notin B, \forall e \in C_B(f) \tag{2}$$

Such a program with a quadratic objective and linear constraints can be solved in polynomial time and a number of practical solvers such as [12] are available.

4 Matroid Intersection

Similar to the case with a single matroid, we need to derive a necessary and sufficient condition for a common basis B of two matroids to be δ-optimal. We can establish such an optimality condition with the help of an exchange graph associated with the basis B and matroids $\mathbb{M}_1$ and $\mathbb{M}_2$.

▶ **Definition 6** (Exchange Graph). Given two matroids $\mathbb{M}_1 = (X, \mathcal{I}_1)$ and $\mathbb{M}_2 = (X, \mathcal{I}_2)$, a weight function $w : X \to \mathbb{R}^+$, and a common basis B, an *exchange* graph is a directed, bipartite graph $G = (V, A)$ with a length function l on edges that is defined as follows.

$$V = B \cup X \setminus B \tag{3}$$

$$A = A_1 \cup A_2 \tag{4}$$

$$A_1 = \{(x, y) | x \in B, y \in X \setminus B, B - \{x\} + \{y\} \in \mathcal{I}_1\} \tag{5}$$

$$A_2 = \{(y, x) | x \in B, y \in X \setminus B, B - \{x\} + \{y\} \in \mathcal{I}_2\} \tag{6}$$

$$l(s) = \begin{cases} w(x) & \text{if } s = (x, y) \in A_1 \\ -w(y) & \text{if } s = (y, x) \in A_2 \end{cases} \tag{7}$$

The above graph captures the exchange operations that can be performed. An edge (e, f) implies that deleting e and adding f to B preserves independence w.r.t matroid $\mathbb{M}_1$ and similarly for the other direction. As the graph is bipartite, every cycle is of even length - a cycle $C = (x_1, y_1, x_2, y_2, \ldots x_k, y_k, x_1)$ corresponds to constructing a set $B'' = B - \{x_1, x_2, \ldots x_k\} \cup \{y_1, y_2, \ldots, y_k\}$. Further

$$w(B'') = w(B) - \sum_{i=1}^{k} w(x_i) + \sum_{i=1}^{k} w(y_i) = w(B) - l(C)$$

where $l(C) = \sum_{e \in C} l(e)$ is the sum of lengths of edges in the cycle C. We are now in a position to present the δ-optimality condition of B in terms of the exchange graph. Fujishige [11] shows the following lemma for the case of $\delta = 0$. We include the extended proof for general δ margin here for completeness. It is important to note that while there are other optimality conditions for matroid intersection such as the weight decomposition theorem by Frank [10], these conditions do not easily generalize for non-zero δ.

▶ **Lemma 7** (Matroid Intersection δ-optimality condition)**.** *The given common basis B is δ-optimal if and only if the exchange graph G contains no directed cycle C such that $\sum_{e \in C} l(e) \leq \delta$.*

Proof. We'll refer to two well-known lemmas [21] regarding the relationship between bases of a matroid and matchings in the exchange graph. Let $G_1 = (V, A_1)$ and $G_2 = (V, A_2)$ be the subgraphs of G induced by the two matroids respectively. Further for $B' \subset X$, let $G(B, B')$ denote the subgraph induced on the G by the vertex sets $B \setminus B'$ and $B' \setminus B$.

▶ **Lemma 8** (Corollary 39.12a in [21])**.** *If B' is a basis of matroid $\mathbb{M}_1$ [$\mathbb{M}_2$], then $G_1(B, B')$ [$G_2(B, B')$] contains a perfect matching.*

▶ **Lemma 9** (Corollary 39.13 in [21])**.** *For $B' \subseteq X$, if $G_1(B, B')[G_2(B, B')]$ has a* unique *perfect matching, then B' is a basis of $\mathbb{M}_1$ [$\mathbb{M}_2$].*

Sufficiency: This is the easy direction. Let B' be any common basis other than B. Applying Lemma 8, we know that $G(B, B')$ has two perfect matchings (one each in $G_1(B, B')$ and $G_2(B, B')$). Union of these two perfect matchings yields a collection of cycles $\mathcal{C}$. Further, by construction, by traversing these cycles, one can transform $B \to B'$ and hence, we have $w(B') = w(B) - \sum_{C \in \mathcal{C}} l(C)$. Therefore, since we have $l(C) > \delta$ for all cycles, we are guaranteed that $w(B') < w(B) - \delta$ as desired.

Necessity: Ideally, we would like to say that every cycle in G leads to a swapping such that the set so obtained is also independent in both the matroids. This would immediately imply that a cycle of small length would lead to a common basis B' which is not much smaller than B.

However, the presence of a cycle simply implies the presence of a perfect matching (one in each direction) which may not be *unique*. For example, Figure 2 shows an instance of an arborescence problem (left), and the associated exchange graph (right). Here G contains a cycle a-x-b-y-a which leads to a new set x, y, c which is not an arborescence.

In the previous example, observe that if the cycle a-x-b-y-a were to have small weight, that would imply that at least one of a-y-a or b-x-b cycles too has small weight both of which lead to a feasible solution. This observation motivates us to look at the *smallest* cycle of weight less than δ and hope that it does induce an *unique* perfect matching.

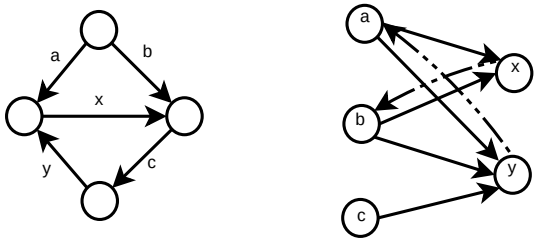


Figure 2 Instance showing every cycle in G need not lead to a common basis.

Suppose that the graph has a cycle having weight less than δ. Let C be the smallest (in terms of number of arcs) such cycle. Look at the graph induced by the vertex set of the cycle. We claim that this induced subgraph has a unique perfect matching (one in each direction). Here we prove the claim for one direction. C being an even cycle trivially contains a perfect matching M from B-side to $X \setminus B$-side. Suppose there exists another perfect matching M'. For every edge (x, y) in $M' \setminus M$, the edge along with the path between y and x in C cause a cycle. Further, each such cycle is smaller (number of edges) than C.

Let $\bar{M}$ denote the matching M with edge directions reversed. The union of M' and $\bar{M}$ now forms a collection of cycles. Consider any such cycle D. WLOG let the cycle be $(x_0, y_0, x_1, y_1, \ldots, x_k, y_k, x_0)$ such that the (x_{i+1}, y_i) are edges in M (i.e. $(y_i, x_{i+1}) \in \bar{M}$) and $(x_i, y_i) \in M'$. [All arithmetic is modulo $k+1$]. We'll now be interested in the length of the path between these vertices in the original cycle C. Let C_i denote the cycle formed by the edge (x_i, y_i) and the path between y_i and x_i in C. We have,

$$l(C_i) = l(C) - l(\text{Path from } x_i \text{ to } y_i \text{ in } C) + l((x_i, y_i))$$

Since $(x_i, y_{i-1}) \in M$,

$$l(\text{Path from } x_i \text{ to } y_i \text{ in } C) = l((x_i, y_{i-1})) + l(\text{Path from } y_{i-1} \text{ to } y_i \text{ in } C)$$

Further since by construction $l((x, y_i)) = l((x, y_j))(= \pm w(x))$, we have

$$l(C_i) = l(C) - l(\text{Path from } y_{i-1} \text{ to } y_i \text{ in } C)$$

Let $P_{i-1 \to i}$ denote this path. Summing over all (x_i, y_i) edges in D, we get

$$\begin{aligned}\sum_{i=0}^{k} l(C_i) &= kl(C) - (l(P_{k\to 0}) + l(P_{0\to 1}) + \ldots + l(P_{k-1\to k})) \\ &= kl(C) - k'l(C)\end{aligned}$$

$\uparrow$ Since we start from y_k, go around the C and reach y_k back

$$\begin{aligned}&= k''l(C) \\ &< k''\delta\end{aligned}$$

The sum of k weights is less than $k''\delta$ with $k'' < k$, which implies

$$\exists C_i, \text{such that } l(C_i) < \delta$$

But this is a contradiction since C was the smallest cycle having weight less than δ. Hence, the perfect matching M is unique. Similarly, the perfect matching induced by C in

the other direction too is unique. Applying Lemma 9 successively on both sides, we know that B' obtained by exchanging as per C is a common basis for both matroids. Further, we have

$$w(B') = w(B) - l(C)$$
$$w(B') > w(B) - \delta$$

Hence we have proved that if G has a cycle with small weight, then B is not δ-optimal, thus proving the necessity of the claim. ◀

4.1 Lower bounding cycles

In order to use Lemma 7 to solve the inverse matroid intersection problem efficiently using quadratic programming, we need a way to formulate this condition as a polynomial number of linear constraints. We now explore a technique to express the condition that a given graph has no small (of length less than δ) cycles concisely. Say we are given a directed graph $G = (V, A)$ and our task is to assign edge-lengths so that all cycles in G have weight at least δ. Letting the edge-lengths to be variables, the feasible region in this case is unbounded and is defined by a constraint for every cycle in G, i.e. we have the region R_1 in m dimensions defined by

$\mathbf{R_1}$:

$$\sum_{e \in C} l_e \geq \delta \qquad \text{For all cycles C} \tag{8}$$

Of course, this formulation has an exponential number of constraints. Although the ellipsoid algorithm can be used to solve the quadratic program in polynomial time, it is often too slow for practical use. We now show that we can obtain a concise extended formulation by adding a few extra variables.

Suppose we have variables d_{xy} representing the shortest distance between vertices x and y. In this case, the graph has no cycle of weight less than δ if and only if $d_{xx} \geq \delta$ for all vertices x (assume $d_{xx} = \infty$, if x is not in any cycle). Consider the region R_2 in $m + n^2$ dimensions:

$\mathbf{R_2}$:

$$d_{xy} \leq l_{(xy)} \qquad \text{For all } (x, y) \in A \tag{9}$$
$$d_{xz} \leq d_{xy} + l_{(yz)} \qquad \text{For all } x, z \in V \text{ and } y \text{ s.t. } (y, z) \in A \tag{10}$$
$$d_{xx} \geq \delta \qquad \text{For all } x \in V \tag{11}$$

Constraints (9) and (10) enforce triangle inequality, and (11) enforce the condition that all cycles are large. We now prove that optimizing any function of l over R_1 is equivalent to optimizing the same over R_2.

▶ **Lemma 10.** *R_1 is identical to the projection of R_2 on the m dimensions corresponding to the edge-lengths.*

Proof.

$\mathbf{R_1} \subseteq$ Projection($\mathbf{R_2}$): Let $l : E \to \mathbb{R}$ denote a point in R_1. Since the constraints (9) and (10) are always valid for a *true* distance function, let $d : V \times V \to \mathbb{R}$ denote the actual distance function in the graph induced by l. Such a d definitely satisfies constraints (9) and (10). Additionally, for all vertices x belonging to some cycle, since all cycles under l have weight at least δ, we have $d_{xx} \geq \delta$. For a vertex x which does not belong to any cycle, one can simply set $d_{xx} = \infty$.

Projection($\mathbf{R_2}$) $\subseteq \mathbf{R_1}$: Consider a point in R_2. We now have the lengths of edges l_e as well as some d_{xy} values. Consider any cycle $C = (x_1, x_2, \ldots, x_k, x_1)$ in the graph. Applying constraint (10) repeatedly we get

$$d_{x_1x_1} \leq l_{(x_1x_2)} + l_{(x_2x_3)} + \ldots + l_{(x_{k-1}x_k)} + l_{(x_kx_1)} \tag{12}$$

and also by constraint (11), we have

$$d_{x_1x_1} \geq \delta \tag{13}$$

Hence we have, $l_{(x_1x_2)} + l_{(x_2x_3)} + \ldots + l_{(x_{k-1}x_k)} + l_{(x_kx_1)} \geq \delta$, i.e. $\sum_{e \in C} l_e \geq \delta$ which means that the l_e values are feasible in R_1. ◀

Hence, optimizing any function of the l_e variables over R_1 is equivalent to optimizing it over R_2. However, R_2 has only $m + mn + n$ constraints and $n^2 + m$ variables.

4.2 Putting it together

Lemmas 7 and 10 suggest a way to solve the δ-margin inverse matroid intersection problem. As per the requirements of Lemma 7, given the two matroids and the common basis B, construct the exchange graph $G = (V, A = A_1 \cup A_2)$. Let $w : X \to \mathbb{R}^+$ be the original weight function and let w' be the new weight function which we desire. If l is the arc lengths of G, according to the construction of Lemma 7, $l_{xy} = w'(x)$ and $l_{yx} = -w'(y)$ where $x \in B, y \in S \setminus B$. Further, the objective that we minimize is the L_2 norm of $w - w'$. We can now add these additional constraints and the objective to the region R_2 as per Lemma 10 to obtain the minimum change on the weights of elements so that the exchange graph has no small cycles and hence B is $\delta-$optimal.

$$\min_{w'} \sum_{e \in X} (w'(e) - w(e))^2 \qquad \textbf{subj. to:} \tag{14}$$

$$l_{xy} = w'(x), \qquad \forall (x, y) \in A_1 \tag{15}$$

$$l_{yx} = -w'(y), \qquad \forall (y, x) \in A_2 \tag{16}$$

$$d_{xy} \leq l_{xy}, \qquad \forall (x, y) \in A \tag{17}$$

$$d_{xz} \leq d_{xy} + l_{yz}, \qquad \forall x, z \in V, \forall (y, z) \in A \tag{18}$$

$$d_{xx} \geq \delta, \qquad \forall x \in V \tag{19}$$

4.3 Maximum Weight Arborescence

Given a directed graph, a r-arborescence (also known as a branching) is the directed analogue of a spanning tree and is defined as a set of edges T spanning all vertices such that every vertex (except r) has exactly one incoming edge in T. It is well known that an arborescence in a directed graph is a basis in the intersection of a graphical matroid and a partition matroid. We analyze the complexity of the above technique for the special case of maximum weight arborescence. Let G denote the graph in question having n vertices and m edges.

The exchange graph G_{ex} has a vertex for every edge of G, i.e., $n_{ex} = m$. The bipartition of G_{ex} is such that we have components of size n and $m - n$ respectively. Hence we have $m_{ex} = O(mn)$. As seen in Section 4.1, we use $O(n_{ex}^2)$ variables and $O(m_{ex}n_{ex})$ contraints. Thus, putting it all together, we have a quadratic program with $O(m^2)$ variables and $O(m^2n)$ constraints.

The inverse maximum weight arborescence problem is important as it can used as a subroutine in the online learning for dependency parsing [19]. The dependency parse tree of a sentence can be represented as an arborescence over a graph consisting of every word in the sentence as a node. In full version of the paper [7], we show experimental results for dependency parsing using our framework.

4.3.1 Shortest $s-t$ paths

Given a weighted graph $G = (V, E, w)$, a path P between terminals s and t, and a margin δ, the inverse shortest s-t path problem is to find a minimum perturbation to w (minimizing the L_2 norm) so that P is shorter than all other paths between s and t by at least δ under the new weight function. As shown by [26], the inverse shortest s-t path problem can be reduced to the inverse arborescence problem. Let G' be G augmented by adding zero weight edges from t to all other vertices. It can be easily observed that P is the shortest s-t path in G if and only if P and a subset of the zero weight edges form the minimum weight s-arborescence of G'. Thus we can use an algorithm for inverse minimum weight arborescence to solve the inverse shortest path problem.[1]

5 Perfect Matchings in Bipartite Graphs

For the bipartite maximum weight perfect matching inverse problem, the previous technique yields a quadratic program having $O(m^2)$ variables and $O(m^2)$ constraints as the exchange graph is sparse. In this section we show that we can in fact obtain more concise formulations. Recall that for a given edge weighted, bipartite graph $G = (X \cup Y, E, w)$, and a perfect matching M, an alternating cycle is a cycle in G in which edges alternate between those that belong to M and those that do not. An alternating cycle C is called δ-augmenting, if $\sum_{e \in C \cap M} w(e) < \sum_{e \in C \setminus M} w(e) + \delta$. The following characterization of a δ-optimal perfect matching is well known.

▶ **Lemma 11.** *A perfect matching M is δ-optimal if and only if the graph contains no δ-augmenting cycles.*

The central idea is to construct a directed graph H on just the nodes of X such that any directed cycle in H will correspond to an alternating cycle in G (w.r.t to the matching M) and vice versa. We construct $H = (X, A)$ to be a directed graph such that $(x, z) \in A$ if and only if $\exists y \in Y$ such that $(x, y) \in M$ and $(y, z) \in E$; further let $l(x, z) = w(x, y) - w(y, z)$. Figure 3 shows an example of this construction.

▶ **Proposition 12.** *The auxiliary graph H has a directed cycle of length less than δ if and only if G has a δ-augmenting alternating cycle.*

Proof.

If: Let $C = (x_0, y_0, x_1, y_1, \ldots, x_k, y_k, x_0)$ be a δ-augmenting cycle in G where all $(x_i, y_i) \in M$. By construction, H has a cycle $C' = (x_0, x_1, \ldots, x_k, x_0)$ and $l(C') = \sum_{i=0}^{k}(w(x_i, y_i) - w(y_i, x_{i+1}))$ (modulo $k+1$) $= \sum_{e \in C \cap M} w(e) - \sum_{e \in C \setminus M} w(e) < \delta$.

[1] Inverse minimum weight arborescence problem can be solved similar to the inverse maximum weight arborescence problem.

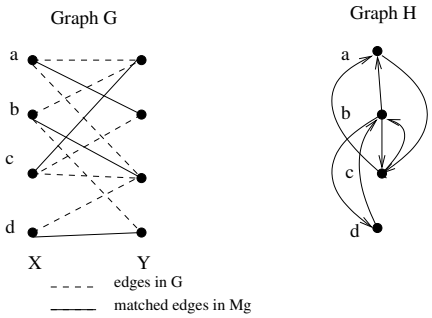


Figure 3 Example to show construction of H from a bipartite graph G and matching M.

Only If: Let $C = (x_0, x_1, \ldots, x_k, x_0)$ be a cycle in H with $l(C) < \delta$. By construction, $\exists$ cycle $C' = (x_0, y_0, x_1, y_1, \ldots, x_k, y_k, x_0)$ in G. Now, $l(C) = \sum_{i=0}^{k}(w(x_i, y_i) - w(y_i, x_{i+1}))$ (modulo $k+1$) $= \sum_{e \in C' \cap M} w(e) - \sum_{e \in C' \setminus M} w(e)$. Thus C' is a δ-augmenting cycle in G. ◀

Using Lemma 11 and Proposition 12 along with Lemma 10, we can formulate the inverse perfect matching problem as a quadratic program having $O(n^2)$ variables and $O(mn)$ constraints.

6 Application: Online learning for structured prediction

In this section, we present a framework for online learning using inverse combinatorial optimization. The structured prediction task is to predict a discrete combinatorial structure (such as an arborescence) given a structured input (such as a graph). The learning task is to learn model parameters so that solving a combinatorial optimization problem on the input instance would return the desired output structure. Structured prediction is extensively used in natural language processing tasks such as obtaining parse trees of a sentence, or automatic language translation.

In the online learning setting, we are presented with a set of T training samples. These consist of an input x_t (for instance, a sentence) and an output y_t (for instance, a syntactic analysis of this sentence described as an arborescence on a graph over the words in the sentence [25, 19]). Each edge in this graph is parameterized by a set of F features that, for instance, indicate how likely one word is to be the subject of another. Thus, each training sample is a pair (x_t, y_t) where x_t is a graph parameterized by features on edges, and y_t is the desired output sub-structure (such as a spanning tree, or an arborescence, or a matching depending on the application). The task is to learn a vector (of length F) of parameters $\boldsymbol{\theta}$ such that when edge weights are computed as inner products between the $\boldsymbol{\theta}$ and the edge's features, the output obtained by computing an optimal sub-structure (spanning tree, etc.) is the desired output with some margin.

Algorithm 1 describes the generic online learning framework for structured prediction. It is parameterized by an user-defined loss function $\ell(y_t, \hat{y})$ that specifies the loss incurred by the prediction $\hat{y}$ with respect to the training solution y_t. Algorithm 1 is an adaptation of the Passive-Aggressive MIRA algorithm [4] for structured prediction.

Note that the minimization problem solved for each training sample is exactly δ-inverse optimization where we minimize the perturbations to the feature parameters instead of the edge weights. In this framework, the different inverse optimization problems we considered have applications for different structured predictions. For example, maximum weight arbor-

$\boldsymbol{\theta}_1 = \mathbf{0}$
for $t = 1$ *to* T **do**
  Obtain training example x_t, y_t
  $w \leftarrow$ weight function s.t. $w(e) = \boldsymbol{\theta}_t \cdot \boldsymbol{f_e}$ where $\boldsymbol{f_e}$ is feature vector of edge e
  $\hat{y} \leftarrow$ optimal sub-structure for graph x_t under weights w
  Suffer loss $\delta_t = \ell(y_t, \hat{y})$
  Update $\boldsymbol{\theta}_{t+1} = \text{argmin}_{\boldsymbol{\theta}'} ||\boldsymbol{\theta}' - \boldsymbol{\theta}_t||_2^2$ **such that**
    $w' \leftarrow$ weight function s.t. $w'(e) = \boldsymbol{\theta}' \cdot \boldsymbol{f_e}$ where $\boldsymbol{f_e}$ is feature vector of edge e
    y_t is the δ_t-optimal sub-structure for graph x_t under weights w'
end
Return $\boldsymbol{\theta}_{T+1}$

Algorithm 1: Generic online learning framework.

escences are used to predict the parse tree of a sentence [25, 19], while maximum weight matchings are used for language translation and word alignments [23].

Since we have shown that we can efficiently solve the inverse optimization problems for a variety of combinatorial structures, we can extend the error bounds of the MIRA algorithm [4] to work for learning the corresponding structured prediction models. In this section, we present both convergence results and loss bounds for our generic online learning framework. The proofs for these bounds closely follow those in Crammer's Ph. D. dissertation [4] and are included in the full version. The statement of the convergence result depends on a set of dual variables obtained from the optimization problem in the "Update" step of Algorithm 1. This implicitly encodes constraints over all possible outputs; we denote the dual variable for output y on the t^{th} example by α_y^t. We can show that the cumulative sum of these dual variables is bounded by a constant independent of T, which implies convergence of the learning algorithm.

▶ Theorem 1 (Convergence). Let $\{(x_t, y_t)\}_{t=1}^T$ be a sequence of structured examples. Let $\boldsymbol{\theta}^*$ be any vector that separates the data with a positive margin $\delta^* > 0$. Assume the loss function is upper bounded: $\ell(y_t, \hat{y}) \leq A$. Then the cumulative sum of coefficients is upper bounded by:

$$\sum_{t=1}^{T} \sum_{y \in \mathcal{Y}^t} \alpha_y^t \leq 2A \left(\frac{||\boldsymbol{\theta}^*||}{\delta^*} \right)^2. \tag{20}$$

However, it is not enough to show that the algorithm converges: it could converge to a useless solution! We wish to show that in the process of learning it does not make too many errors. In particular, we show that Algorithm 1 incurs a total hinge loss bounded by a constant also independent of T, which implies that at some point it has exactly solved the learning problem.

▶ Theorem 2 (Total Loss). Under the same assumptions as above, assume further that the norm of the examples are bounded by R. Then, the cumulative hinge loss ($\mathcal{H}_{\delta_t}$) suffered by the algorithm over T trials is bounded by:

$$\sum_{t=1}^{T} \mathcal{H}_{\delta_t}(\theta_t, (x_t, y_t)) \leq 8A \left(\frac{R\,||\boldsymbol{\theta}^*||}{\delta^*} \right)^2. \tag{21}$$

References

1 David Chiang. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 2012.

2 Y.J. Chu and T.H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

3 Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

4 Koby Crammer. *Online Learning of Complex Categorical Problems*. PhD thesis, Hebrew University of Jerusalem, 2004.

5 Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research (JMLR)*, 2006.

6 Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research (JMLR)*, 2003.

7 Hal Daumé III, Samir Khuller, Manish Purohit, and Gregory Sanders. On correcting inputs: Inverse optimization for online structured prediction. *CoRR*, 2015. `http://arxiv.org/abs/1510.03130`.

8 Mauro Dell'Amico, Francesco Maffioli, and Federico Malucelli. The base-matroid and inverse combinatorial optimization problems. *Discrete applied mathematics*, 128(2):337–353, 2003.

9 J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

10 András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, 1981.

11 Satoru Fujishige. A primal approach to the independent assignment problem. *Journal of the Operations Research Society of Japan*, 20(1):1–15, 1977.

12 Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2012.

13 Clemens Heuberger. Inverse combinatorial optimization: A survey on problems, methods, and results. *Journal of Combinatorial Optimization*, 8(3):329–361, 2004.

14 Jyrki Kivinen and Manfred Warmuth. Exponentiated gradient versus gradient descent for linear predictors. In *Symposium on the Theory of Computing (STOC)*, 1995.

15 John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

16 Cai Mao-Cheng and Yanjun Li. Inverse matroid intersection problem. *Mathematical Methods of Operations Research*, 45(2):235–243, 1997.

17 David McAllester, Michael Collins, and Fernando Pereira. Case-factor diagrams for structured probabilistic modeling. In *Proceedings of the Converence on Uncertainty in Artificial Intelligence (UAI)*, 2004.

18 Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, 2005.

19 Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Joint Conference on Human Language Technology Conference and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

20 Vasin Punyakanok and Dan Roth. The use of classifiers in sequential inference. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

21 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Verlag, 2003.

22 Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 897–904, 2005.

23 Ben Taskar, Simon Lacoste-Julien, and Dan Klein. A discriminative matching approach to word alignment. In *Proceedings of EMNLP 2005*, 2005.

24 Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasmine Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, Sep 2005.

25 Daniel Zeman. A statistical approach to parsing of czech. *Prague Bulletin of Mathematical Linguistics*, 69:29œ37, 1998.

26 Hu Zhiquan and Liu Zhenhong. A strongly polynomial algorithm for the inverse shortest arborescence problem. *Discrete applied mathematics*, 82(1):135–154, 1998.

Dynamic Sketching for Graph Optimization Problems with Applications to Cut-Preserving Sketches*

Sepehr Assadi†, Sanjeev Khanna†, Yang Li†, and Val Tannen‡

University of Pennsylvania, Philadelphia, US
{sassadi,sanjeev,yangli2,val}@cis.upenn.edu

Abstract

In this paper, we introduce a new model for sublinear algorithms called *dynamic sketching*. In this model, the underlying data is partitioned into a large *static* part and a small *dynamic* part and the goal is to compute a summary of the static part (i.e, a *sketch*) such that given any *update* for the dynamic part, one can combine it with the sketch to compute a given function. We say that a sketch is *compact* if its size is bounded by a polynomial function of the length of the dynamic data, (essentially) independent of the size of the static part.

A graph optimization problem P in this model is defined as follows. The input is a graph $G(V, E)$ and a set $T \subseteq V$ of k terminals; the edges between the terminals are the dynamic part and the other edges in G are the static part. The goal is to summarize the graph G into a compact sketch (of size $\text{poly}(k)$) such that given any set Q of edges between the terminals, one can answer the problem P for the graph obtained by inserting all edges in Q to G, using only the sketch.

We study the fundamental problem of computing a maximum matching and prove tight bounds on the sketch size. In particular, we show that there exists a (compact) dynamic sketch of size $O(k^2)$ for the matching problem and any such sketch has to be of size $\Omega(k^2)$. Our sketch for matchings can be further used to derive compact dynamic sketches for other fundamental graph problems involving cuts and connectivities. Interestingly, our sketch for matchings can also be used to give an elementary construction of a *cut-preserving vertex sparsifier* with space $O(kC^2)$ for k-terminal graphs, which matches the best known upper bound; here C is the total capacity of the edges incident on the terminals. Additionally, we give an improved lower bound (in terms of C) of $\Omega(C/\log C)$ on size of cut-preserving vertex sparsifiers, and establish that progress on dynamic sketching of the s-t max-flow problem (either upper bound or lower bound) immediately leads to better bounds for size of cut-preserving vertex sparsifiers.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity

Keywords and phrases Small-space Algorithms, Maximum Matchings, Vertex Sparsifiers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.52

1 Introduction

Massive data sets are arising more and more frequently in many application domains. Traditional gold standards of computational efficiency, namely, linear-time and linear-space, no longer seem sufficient for managing and analyzing such massive data sets. As a result, a

* The full version of the paper can be found at [5].
† Supported in part by National Science Foundation grants CCF-1116961, CCF-1552909, and IIS-1447470.
‡ Supported in part by National Science Foundation grants IIS 1217798 and 1302212.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 52–68

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

beautiful new area of sublinear algorithms has developed over the past two decades – these are algorithms whose resource requirements are substantially smaller than the size of the input on which they operate. A rich theory of sublinear algorithms has emerged, and has brought remarkable new insights into combinatorial structure of well-studied optimization problems (see, for instance, the surveys [23, 25, 27], and references therein).

In recent years, graph optimization problems have received a lot of attention in the study of sublinear algorithms in various models, and the streaming model of computation is one of the most popular examples. In the streaming model, an algorithm is presented with a stream of edge insertions and deletions and is required to give an answer to a pre-specified graph problem at the end of the stream. Unfortunately, for many fundamental graph problems, no small space streaming algorithm is possible. For instance, [10] showed that determining whether or not there is a path from a specific vertex s to a specified vertex t in a directed graph requires $\Omega(n^2)$ space even for streams with only edge insertions; here n denotes the number of vertices in the input graph. This immediately implies that computing the length of the s-t shortest path, the value of the minimum cut between s and t, or the edge/vertex connectivity between s and t also requires $\Omega(n^2)$ space since the output of these problems is non-zero only when there is a path from s to t. The same lower bound is also obtained for computing the size of the maximum matching [10]. In fact, most of recent works for graph problem focus on *approximation algorithms* developed under the *semi-streaming* model introduced in [10], where an algorithm is allowed to output an approximate answer while using space linear in n. But is there hope left for *exact* sublinear algorithms? More specifically, is there a non-trivial model where sublinear algorithms are achievable for outputting exact answers for fundamental graph problems like matchings, connectivities, cuts, etc.?

In this paper, we explore this direction by considering the case where the input graph only undergoes *local changes*, and study how local changes influence the solutions of several fundamental graph problems. The goal is to exploit the locality of these updates and compress the rest of the graph into a small-size *sketch* that is able to answer *queries* regarding a specific problem (e.g. the s-t edge connectivity problem) for every possible local changes made to the graph. We introduce a model in this spirit and in the rest of this section, we formally define the model, discuss the connection to existing models, and summarize our results.

1.1 The Dynamic Sketching Model

We define the *dynamic sketching model*, where algorithms are required to construct data structures (called sketches) that are composable with *local updates* to the underlying data. Specifically, for graph problems in the dynamic sketching model, we consider the following setup (see Section 1.1 in the full version [5] for a more general definition which is not restricted to graph problems). Given a graph optimization problem P, an input graph $G(V, E)$ on n vertices with k vertices identified as *terminals* $T = \{q_1, \ldots, q_k\}$, the goal of k-dynamic sketching for P is to construct a sketch Γ such that given any possible subset of the edges between the terminals (a *query*), we can solve the problem P using only the information contained in the sketch Γ. Formally,

▶ **Definition 1.** Given a graph-theoretic problem P, a k-dynamic sketching scheme for P is a pair of algorithms with the following properties.

- **(i)** A **compression algorithm** that given any input graph $G(V, E)$ with a set T of k terminals, outputs a data structure Γ (i.e, a dynamic sketch).
- **(ii)** An **extraction algorithm** that given any subset of the edges between the terminals, i.e, a **query** Q, and the sketch Γ, outputs the answer to the problem P for the graph, denoted by G^Q, obtained by inserting all edges in Q to G (without further access to G).

We allow both compression and extraction algorithms to be randomized and err with some small probability. Furthermore, we say a sketching scheme is *compact* if it constructs dynamic sketches of size $\text{poly}(k)$, where the size of a sketch is measured by the number of machine words of length $O(\log n)$.

We should note right away that of course *not* every graph problem admits a compact dynamic sketch. For example, one can show that any dynamic sketch for the *maximum clique* problem or the *minimum vertex cover* problem requires $\min\{\Omega(n), 2^{\Omega(k)}\}$ space (see the full version of the paper [5], Section 6).

1.2 Connection to Existing Models

Streaming. Any single-pass streaming algorithm with space requirement s can be used as a dynamic sketching scheme with a sketch of size s: run the streaming algorithm on graph $G(V, E)$ for the static data and store the state of the algorithm as the sketch; continue running the algorithm using the stored state when the dynamic data is presented. However, note that a streaming algorithm directly gives a *compact* scheme only when the space requirement is logarithmic in n, which, as we just discussed, is not the case for nearly all fundamental graph optimization problems. In the following, we use the s-t shortest path problem as an example to elaborate the distinction between the two models. Our results in Section 2, illustrates a similar distinction for the case of the maximum matching problem.

As we already mentioned, outputting the length of the s-t shortest path requires $\Omega(n^2)$ space in the streaming model. We now give a simple dynamic sketching scheme for the s-t shortest problem with a sketch of size $O(k^2)$. The input to the s-t shortest path problem in the k-dynamic sketching model is a weighted graph G, a set T of terminals, and two designated vertices s and t. Without loss of generality, we can assume s and t are terminals; otherwise we can add them to the set of terminals and record their edges to the other terminals in $O(k)$ space. The compression algorithm creates a graph H with $V(H) = T$, where for any pair of terminals q_i and q_j, a directed edge from q_i to q_j is added to H with weight equal to the weight of a shortest path from q_i to q_j in G. The size of H is $O(k^2)$. To obtain the answer for each query Q, the extraction algorithm adds the edges in Q to H, building a small graph H^Q, and compute the shortest path from s to t in H^Q. It is easy to see that the weights of the shortest paths between s and t in H^Q and G^Q are equal, thus H can be used as a dynamic sketch for the s-t shortest path problem.

Linear sketches. A very strong notion of sketching for handling arbitrary changes to the original data is linear sketching, which corresponds to applying a randomized low-dimensional linear transformation to the input data. This allows for compressing the data into a smaller space while (approximately) preserving some desired property of the input. Moreover, composability of these sketches (as they are linear transformations) allow them to handle arbitrary changes to the input data. Linear sketching technique has been successfully applied to various graph problems, mainly involving cuts and connectivity [2, 3, 14] (see also [23] for a survey of such results in dynamic graph streams). However, these results use space that is prohibitively large for dynamic sketching (a linear dependence on n), and typically only yield approximation answers.

Kernelization. Dynamic sketching shares some similarity to *Kernelization* developed in parametrized complexity [17, 16, 11] in the following two aspects. Firstly, the number of terminals k in dynamic sketching may be viewed as a parameter. However, the main difference here is that for dynamic sketching, k is a parameter of the *model*, while for kernelization, the

parameter is usually the size of the solution, which is the property of the input rather than the model. Secondly, since a kernel for an instance of a problem is defined to be an equivalent instance of the same problem with size bounded by a function of a fixed parameter of the problem, both dynamic sketching and kernelization are in the spirit of compression. However, the techniques developed in kernelization do not directly carry over to dynamic sketching for the following two reasons. Firstly, kernelization typically focuses only on static data and secondly, the space target in kernelization (which is different compare to dynamic sketching) is normally polynomial in the parameter (usually the size of the solution to the problem) which could be $\Omega(n)$ in the dynamic sketching model. Finally, it is worth mentioning that there are problems (e.g. *minimum vertex cover*) that admit polynomial size kernels, while it can be shown that the dynamic sketching for these problem require sketches of size $2^{\Omega(k)}$ (see the full version of the paper [5], Section 6).

Provisioning. We should note that dynamic sketching shares some ancestry with *provisioning*, a technique developed by [8] for avoiding repeated expensive computations in what-if analysis, where the input data is formed by k known overlapping subsets of some universe, and the goal is to compress these subsets so as to answer a specific database query when only some of those subsets are presented at run-time. Note that a main distinction between the two model is that in provisioning the dynamic input is neither small nor local.

1.3 Our Results

Maximum matching. The main focus of this paper is on the maximum matching problem [20] in the dynamic sketching model, and its applications to various others problems. We give a dynamic sketching scheme with a sketch of size $O(k^2)$, using a technique based on an algebraic formulation of the matchings introduced by Tutte [29]. At a high level, we store a sketch that computes the rank of the *Tutte matrix* (see Definition 4) of the underlying graph. Since the queries only affect $O(k^2)$ entries of the Tutte matrix, we can compress this matrix using algebraic operations into a few small matrices of dimensions $k \times k$. Storing the small matrices as the sketch and modifying the related entries when a query is presented allows us to compute the rank of the original Tutte matrix, and hence the maximum matching size. Furthermore, we prove that our sketching scheme is optimal in terms of its space requirement (up to a logarithmic factor). In particular, we show that any dynamic sketching scheme for the matching problem has to store a sketch of size $\Omega(k^2)$ bits. We emphasize that the lower bound is information-theoretic; it holds even if the compression and extraction algorithms are computationally unbounded.

Cut-preserving sketches. Interestingly, we discovered that our scheme for matchings can be used to design a *cut-preserving sketch*, which is the information-theoretic version of a *cut-preserving vertex sparsifier* [12, 24, 19]. Given a capacitated graph G (assume all capacities are integers) with a set T of k terminals, a cut-preserving vertex sparsifier (or a sparsifier for short) of G is a graph H with $T \subseteq V(H)$ ($V(H)$ denotes the set of vertices of H) such that for any bipartition S and $T \setminus S$ of terminals, the value of the minimum cut between S and $T \setminus S$ in G is preserved in H. A vertex sparsifier where the stored data is not restricted to be a graph is called a cut-preserving sketch.

In recent years, cut-preserving vertex sparsifiers have been extensively studied (see, for example, [6, 9, 7, 4]). For instance, exact sparsifiers with 2^{2^k} vertices are shown by [12, 15], and sparsifiers with $O(C^3)$ vertices are shown by [17], where C is the total capacity of the edges incident on the terminals. Additionally, the size of any exact sparsifier is shown to be

$2^{\Omega(k)}$ [18, 15]. Cut-preserving sketches are also studied in the literature [4, 18, 16], where the best construction is known to be of size $O(kC^2)$ by [16]. Moreover, the $2^{\Omega(k)}$ lower bound of [18] is also shown to hold for the cut-preserving sketches.

We show that our dynamic sketching scheme for matchings can be used to obtain an elementary construction of a cut-preserving sketch of size $O(kC^2)$ that matches the best known upper bound of [16]. [16] showed that given a graph G and a set of k terminals T, a single *gammoid* can be used to produce a matroid that encodes all *terminal vertex cuts*. The authors then use the result of [22] to show how to obtain a matrix representation of this gammoid with $O(k^2)$ entries of $O(k)$ bits each (see Corollary 3.2 of [16]). Using standard techniques, one can use this sketch for vertex cuts to obtain an sketch for edge cuts (i.e, a cut-preserving sketch) that requires $O(kC^2)$ space. Our construction, on the other hand, uses the connection between matchings and the Tutte matrix followed by a simple reduction from cut-preserving sketches to the maximum matching problem. We believe that the simplicity of this construction and its connection to dynamic sketches for the matching problem is of independent interest and gives further insights into the structure of cut-preserving sketches. Moreover, we prove an improved lower bound (in terms of C) of $\Omega(C/\log C)$ bits on the size of any cut-preserving sketch; prior to our work, the best lower bound in terms of C is $\Omega(C^{\varepsilon})$ for some small constant $\varepsilon > 0$ obtained by [18].

s-t edge-connectivity and s-t maximum flow. As it turns out, any cut preserving sketch can be (almost directly) used to obtain a dynamic sketching scheme for the s-t edge-connectivity problem. However, using our lower bound for cut-preserving sketches, the resulting sketch size for edge-connectivity would be $\Omega(C/\log C)$, where C could be as large as n (hence the sketch is not compact). To obtain a compact sketch for edge-connectivity, we further design a dynamic sketching scheme which directly uses our dynamic sketching scheme for matchings, and obtain compact sketches of size $O(k^4)$. We further establish that cut-preserving sketches are, in fact, more related to the s-t maximum flow problem, in the sense that progress on either upper bound or lower bound on size of dynamic sketches for the s-t maximum flow problem immediately leads to better bounds for size of cut-preserving sketches.

Minimum spanning tree. Finally, we present an $O(k)$-size dynamic sketch for the minimum spanning tree (MST) problem. Our idea for creating a compact dynamic sketch for MST is as follows. First of all, it is easy to see that if we add an edge to a graph, an MST of the resulting graph can be created by adding the edge to an MST of the original graph. Hence, it is sufficient to store an MST H of the original graph as a sketch. But this sketch is of size $\Omega(n)$. We show that H can be compressed into a tree H' such that all leaf nodes are terminals and there are at most $O(k)$ internal nodes in this tree; moreover, for any query Q, the weights of the MSTs in G^Q and H'^Q are equal. Hence, H' can be stored as a dynamic sketch. Due to the space constraints, the proof of this result is deferred entirely to the full version of the paper [5] (see Section 5).

Organization. The rest of the paper is organized as follows. We first introduce our dynamic sketching scheme for the maximum matching problem in Section 2 and prove its optimality in terms of the sketch size. Then, in Section 3.1, we show how to use our sketching scheme for the matching problem to construct a cut-preserving sketch. Next, we provide our improved lower bound on the size of cut-preserving sketches in Section 3.2. We further establish the connection between cut-preserving sketches and s-t edge-connectivity (and s-t maximum

flow) and introduce a compact dynamic sketch for edge connectivity in Section 4. Finally, we conclude the paper with some future directions in Section 5.

Notation. We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. The bold-face upper-case letters represent matrices. A matrix with a 'tilde' on top (e.g. $\widetilde{\mathbf{M}}$) denotes a symbolic matrix, i.e, a matrix containing formal variables. For any prime p, $\mathbb{Z}_p$ denotes the field of integers modulo p.

For any undirected graph G, we use $\nu(G)$ to denote the size of a maximum matching in G. For any directed graph $G(V, E)$, an edge $e = (u, v)$ is directed from u to v, where we say u is the *tail* and v is the *head* of e. For any vertex $v \in V$, $d^+(v)$ (resp. $d^-(v)$) denotes the number of outgoing (resp. incoming) edges of v. For a capacitated graph, $c^+(v)$ (resp. $c^-(v)$) denotes the total capacity of the outgoing (resp. incoming) edges of v. We assume all the capacities are integers and can be stored in a single machine word of size $O(\log n)$.

2 The Maximum Matching Problem

In this section, we provide our results for the maximum matching problem. In particular, we show that,

▶ **Theorem 2.** *For any $0 < \delta < 1$, there exists a randomized k-dynamic sketching scheme for the maximum matching problem with a sketch of size $O(k^2 \log(1/\delta))$, which answers any query correctly with probability at least $1 - \delta$.*

Furthermore, we prove that the sketch size obtained in Theorem 2 is tight (up to an $O(\log n)$ factor). Formally,

▶ **Theorem 3.** *For any $k \geq 2$, any k-dynamic sketching scheme for the maximum matching problem that answers any query correctly with probability at least $2/3$, requires a dynamic sketch of size $\Omega(k^2)$ bits.*

Our sketching scheme for the proof of Theorem 2 relies on an algebraic formulation for the matching problem due to Tutte [29]. In the remainder of this section, we present this algebraic formulation, state our sketching scheme for matchings and proves its correctness and then present our lower bound result.

Algebraic formulation for the matching problem. The following matrix was first introduced by Tutte [29].

▶ **Definition 4** (Tutte matrix [29])**.** Suppose $G(V, E)$ is an undirected graph. The *Tutte matrix* of G is the following *symbolic* matrix $\widetilde{\mathbf{M}}$ of dimension $n \times n$.

$$\widetilde{\mathbf{M}}_{i,j} = \begin{cases} x_{i,j} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{j,i} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

where the $x_{i,j}$ are distinct formal variables.

Lovász [21] established the following result for computing the size of a maximum matching using Tutte matrix (see also [26] for more details on performing the computations over a finite field).

▶ **Lemma 5** ([21, 26]). *Let G be an undirected graph with n vertices and the maximum matching size of $\nu(G)$. For any prime $p > n$, let $\mathbb{Z}_p$ be the field of integers modulo p. Suppose $\widetilde{\mathbf{M}}$ is the Tutte matrix of G and $\mathbf{M}$ is the matrix obtained by evaluating each variable in $\widetilde{\mathbf{M}}$ by a number chosen independently and uniformly at random from $\mathbb{Z}_p$; then:*

$$\Pr\left(\text{rank}(\mathbf{M}) = 2\nu(G)\right) \geq 1 - \frac{n}{p}$$

Note that the computation of $\text{rank}(\mathbf{M})$ *is also done over the field $\mathbb{Z}_p$.*

2.1 An $O(k^2)$ size upper bound

In this section, we provide our k-dynamic sketching scheme for the maximum matching problem and prove Theorem 2.

Notation. Suppose the input is an undirected graph $G(V, E)$ with a set $T = \{q_1, \ldots, q_k\}$ of k terminals. Let p be any prime of magnitude $\Theta(n/\delta)$; we perform the algebraic computations in the field $\mathbb{Z}_p$. Let $\widetilde{\mathbf{M}}$ be the Tutte matrix of the graph obtained by adding all edges between the terminals to G, where the first k rows and k columns correspond to the vertices in T. We decompose $\widetilde{\mathbf{M}}$ into four sub matrices $\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}, \widetilde{\mathbf{C}}$, and $\widetilde{\mathbf{D}}$ as follows:

$$\widetilde{\mathbf{M}} = \begin{bmatrix} \widetilde{\mathbf{A}}_{k\times k} & \widetilde{\mathbf{B}}_{k\times(n-k)} \\ \widetilde{\mathbf{C}}_{(n-k)\times k} & \widetilde{\mathbf{D}}_{(n-k)\times(n-k)} \end{bmatrix}$$

Compression algorithm: The compression algorithm consists of 4 steps. Each of them performs a simple algebraic manipulation on the Tutte matrix $\widetilde{\mathbf{M}}$.

Step 1. For each non-zero entry of $\widetilde{\mathbf{M}}$ that corresponds to an edge in G (i.e., not between the terminals), assign an integer chosen uniformly at random from $\mathbb{Z}_p$. Denote the resulting matrix by,

$$\widetilde{\mathbf{M}}_1 = \begin{bmatrix} \widetilde{\mathbf{A}}_{k\times k} & \mathbf{B}_{k\times(n-k)} \\ \mathbf{C}_{(n-k)\times k} & \mathbf{D}_{(n-k)\times(n-k)} \end{bmatrix}$$

Note that except for $\widetilde{\mathbf{A}}$, all sub-matrices in $\widetilde{\mathbf{M}}_1$ are *no longer* symbolic.

Step 2. Let $r = \text{rank}(\mathbf{D})$. Use *elementary row and column operations* to change $\mathbf{D}$ into a *diagonal* matrix $\text{diag}(1, \ldots, 1, 0, \ldots, 0)$ with only r non-zero entries. Note that after this process, matrices $\mathbf{B}$ and $\mathbf{C}$ would also change, but the symbolic matrix $\widetilde{\mathbf{A}}$ remains unchanged. We denote the matrix $\widetilde{\mathbf{M}}_1$ after this process by,

$$\widetilde{\mathbf{M}}_2 = \begin{bmatrix} \widetilde{\mathbf{A}}_{k\times k} & \mathbf{X}_{k\times r} & \mathbf{B}'_{k\times(n-k-r)} \\ \mathbf{Y}_{r\times k} & \mathbf{I}_{r\times r} & \mathbf{0}_{r\times(n-k-r)} \\ \mathbf{C}'_{(n-k-r)\times k} & \mathbf{0}_{(n-k-r)\times r} & \mathbf{0}_{(n-k-r)\times(n-k-r)} \end{bmatrix}$$

Step 3. Use the sub-matrix $\mathbf{I}_{r\times r}$ in $\widetilde{\mathbf{M}}_2$ to zero out the matrix $\mathbf{X}$ by elementary *row* operations. Similarly, zero out $\mathbf{Y}$ by elementary *column* operations. Note that after this process, the matrix $\widetilde{\mathbf{A}}$ would be added by a linear combination of the rows in $\mathbf{Y}$, denoted by $\mathbf{A}'$. Denote the resulting matrix by,

$$\widetilde{\mathbf{M}}_3 = \begin{bmatrix} \widetilde{\mathbf{A}}_{k\times k} + \mathbf{A}'_{k\times k} & \mathbf{0}_{k\times r} & \mathbf{B}'_{k\times(n-k-r)} \\ \mathbf{0}_{r\times k} & \mathbf{I}_{r\times r} & \mathbf{0} \\ \mathbf{C}'_{(n-k-r)\times k} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

Step 4. Consider the matrix $\mathbf{B}'$ in $\widetilde{\mathbf{M}}_3$; pick a maximal set of *linearly independent* columns from $\mathbf{B}'$ (if less than k columns are picked, arbitrarily pick from the remaining columns until having picked k columns), denoted by $\mathbf{B}''_{k\times k}$. Do the same for the matrix $\mathbf{C}'$ (but using linearly independent rows) and create $\mathbf{C}''_{k\times k}$. Finally, pick k^2 numbers from $\mathbb{Z}_p$, independently and uniformly at random and form a matrix of dimension $k \times k$, denoted by $\hat{\mathbf{A}}$. Store the value r (i.e., the rank of $\mathbf{D}$), the matrix $\hat{\mathbf{A}}$, and three $k \times k$ matrices $\mathbf{A}'$, $\mathbf{B}''$ and $\mathbf{C}''$ as the sketch.

Extraction algorithm: Given a query Q, create the matrix $\hat{\mathbf{A}}_Q$ from $\hat{\mathbf{A}}$ by zeroing out every entry that corresponds to an edge *not* in Q. Evaluate $\widetilde{\mathbf{A}}$ by $\hat{\mathbf{A}}_Q$ and obtain a (non-symbolic) matrix $\mathbf{A}$. Construct a matrix $\hat{\mathbf{M}}$ as follows,

$$\hat{\mathbf{M}} = \begin{bmatrix} \mathbf{A}_{k\times k} + \mathbf{A}'_{k\times k} & \mathbf{B}''_{k\times k} \\ \mathbf{C}''_{k\times k} & \mathbf{0}_{k\times k} \end{bmatrix}$$

Return $\left(\text{rank}(\hat{\mathbf{M}}) + r\right)/2$ as the maximum matching size.

We now prove the correctness of this scheme and show that it satisfies the bound given in Theorem 2 and hence prove this theorem.

Proof of Theorem 2. Since the prime p is of magnitude $\Theta(n/\delta)$, any number in $\mathbb{Z}_p$ requires $O(\log(n/\delta)) = O(\log n + \log(1/\delta))$ bits to store, which is at most $O(\log(1/\delta))$ machine words. The compression algorithm stores a number r, which needs $O(\log n)$ bits, four matrices of dimension $k \times k$, where each entry is a number in $\mathbb{Z}_p$ and requires $O(\log(1/\delta))$ space. Therefore, the total sketch size is $O(k^2 \log(1/\delta))$. We now prove the correctness.

We need to show that for each query Q, the extraction algorithm correctly outputs the matching size with probability at least $1 - \delta$. By Lemma 5,

$$\Pr\left(\text{rank}(\mathbf{M}) = 2\nu(G^Q)\right) \geq 1 - \frac{n}{p} \geq 1 - \delta$$

Here $\mathbf{M}$ is the (randomly evaluated) Tutte matrix of the graph obtained by applying the query Q to G, i.e., G^Q. Since the extraction algorithm outputs $\left(\text{rank}(\hat{\mathbf{M}}) + r\right)/2$ as the matching size, it suffices for us to show that $\text{rank}(\hat{\mathbf{M}}) + r = \text{rank}(\mathbf{M})$.

More specifically, the extraction algorithm evaluates $\widetilde{\mathbf{A}}$ by assigning a (pre-selected) random number to each entry that corresponds to an edge in Q, i.e, the matrix $\hat{\mathbf{A}}_Q$. For the sake of analysis, assume this is done before the compression algorithm is executed. Then, at the first step of the compression algorithm, all entries of the matrices $\mathbf{B}, \mathbf{C}, \mathbf{D}$ are randomly and independently evaluated. Combined with evaluating $\widetilde{\mathbf{A}}$ by $\hat{\mathbf{A}}_Q$, the resulting matrix (denoted by $\mathbf{M}_1$) is obtained from randomly and independently evaluating every non-zero entry of the Tutte matrix of the graph G^Q. In other words, it suffices to show that $\text{rank}(\mathbf{M}_1) = \text{rank}(\hat{\mathbf{M}}) + r$.

Since step 2 and step 3 only perform elementary row/column operations on the matrix, the rank does not change. For the matrix $\widetilde{\mathbf{M}}_3$ obtained after step 3, denote by $\mathbf{M}_3$ the matrix after evaluating the $\widetilde{\mathbf{A}}$ part in $\widetilde{\mathbf{M}}_3$. $\mathbf{M}_3$ is non-symbolic and it suffices to prove that $\text{rank}(\mathbf{M}_3) = \text{rank}(\hat{\mathbf{M}}) + r$. Note that after reordering rows and columns of $\mathbf{M}_3$, $\mathbf{M}_3$ can be rewritten as

$$\begin{bmatrix} \mathbf{A}_{k\times k} + \mathbf{A}'_{k\times k} & \mathbf{B}'_{k\times(n-k-r)} & \mathbf{0} \\ \mathbf{C}'_{(n-k-r)\times k} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{r\times r} \end{bmatrix}$$

Therefore, the rank of $\mathbf{M}_3$ is equal to r plus the rank of the following sub-matrix of $\mathbf{M}_3$.

$$\mathbf{M}_4 = \begin{bmatrix} \mathbf{A}_{k\times k} + \mathbf{A}'_{k\times k} & \mathbf{B}'_{k\times(n-k-r)} \\ \mathbf{C}'_{(n-k-r)\times k} & \mathbf{0} \end{bmatrix}$$

We now show that $\mathbf{M}_4$ has the same rank as $\hat{\mathbf{M}}$. Since the matrix $\mathbf{C}''$ (in step 4 of the compression algorithm) contains a maximal set of linearly independent rows of $\mathbf{C}'$, each remaining row of $\mathbf{C}'$ is a linear combination of the rows in $\mathbf{C}''$. Therefore, all remaining rows in $\mathbf{C}'$ can be zero-out using elementary row operations. Hence, the rank of $\mathbf{M}_4$ is equal to the rank of the following matrix

$$\mathbf{M}_5 = \begin{bmatrix} \mathbf{A}_{k\times k} + \mathbf{A}'_{k\times k} & \mathbf{B}'_{k\times(n-k-r)} \\ \mathbf{C}''_{k\times k} & \mathbf{0} \\ \mathbf{0}_{(n-2k-r)\times k} & \mathbf{0} \end{bmatrix}$$

Similarly, using elementary column operations, the sub-matrix $\mathbf{B}'$ in $\mathbf{M}_5$ can be made into $[\mathbf{B}''\ \ \mathbf{0}_{k\times(n-2k-r)}]$ without changing the rank, and the resulting matrix has the same rank as $\hat{\mathbf{M}}$. ◀

2.2 An $\Omega(k^2)$ size lower bound

In this section, we prove an $\Omega(k^2)$ bits lower bound on the sketch size of any k-dynamic sketching scheme for the matching problem, which implies that our space upper bound in Theorem 2 is tight (up to a logarithmic factor). We establish this lower bound by reducing from the MEMBERSHIP problem studied in communication complexity defined as follows.

The MEMBERSHIP Problem

Input: Alice is given a set $S \subseteq [N]$ and Bob is given an element $e^* \in [N]$.
Goal: Alice has to send a message to Bob such that Bob can determine whether $e^* \in S$ or not.

It is well-known that in order for Bob to succeed with probability at least 2/3, Alice has to send a message of size $\Omega(N)$ bits [1], where the probability is taken over the random coin tosses of Alice and Bob.

Reduction. For simplicity, assume N is a perfect square. Given any $S \subseteq [N]$, Alice constructs a graph $G(V, E)$ with a set T of k terminals as follows:

- The vertex set $V = \{u, w\} \cup V_1 \cup V_2 \cup V_3 \cup V_4$, where $|V_i| = \sqrt{N}$ for any $i \leq 4$ and $T = \{u, w\} \cup V_1 \cup V_4$. We will use $v_j^{(i)}$ to denote the j-th vertex in V_i.
- For any $i \in [\sqrt{N}]$, $v_i^{(1)}$ (resp. $v_i^{(3)}$) is connected to $v_i^{(2)}$ (resp. $v_i^{(4)}$); i.e, there is perfect matching between V_1 and V_2 (resp. V_3 and V_4).
- Fix a bijection $\sigma : [N] \mapsto [\sqrt{N}] \times [\sqrt{N}]$; for any element $e \in S$ with $\sigma(e) = (i, j)$, $v_i^{(2)}$ is connected to $v_j^{(3)}$.

Note that in this construction, $n = 4\sqrt{N} + 2$ and $k = 2\sqrt{N} + 2$, and initially there is no edge between the terminals.

Alice constructs this graph, run the compression algorithm of the dynamic sketching scheme on it and sends the sketch to Bob. Let Q be the query in which, for $\sigma(e^*) = (i, j)$, u is connected to $v_i^{(1)}$ and $v_j^{(4)}$ is connected to w. Bob queries the sketch with Q, finds

the maximum matching size in G^Q, and returns $e^* \in [S]$ iff the maximum matching size is $2\sqrt{N}+1$ in G^Q.

The proof of the following lemma can be found in the full version [5] (Lemma 2.2).

▶ **Lemma 6.** *$\nu(G^Q) = 2\sqrt{N}+1$ if and only if $e^* \in S$.*

Theorem 3 now follows from Lemma 6, along with the lower bound of $\Omega(N) = \Omega(k^2)$ on the communication complexity of the MEMBERSHIP problem.

3 Cut-Preserving Sketches

We establish a connection between k-dynamic sketching schemes for the maximum matching problem and cut-preserving sketches. In particular, we use our dynamic sketching scheme for the matching problem in Section 2 to design an exact cut-preserving sketch (i.e, an information-theoretic vertex sparsifier) with size $O(kC^2)$, where C is the total capacity of the edges incident on the terminals. This matches the best known upper bound on the space requirement of cut-preserving sketches. We further provide an improved lower bound of $\Omega(C/\log C)$ on the size of any cut-preserving sketch. Throughout this section, we will use the term *bipartition cut* to refer to a cut between a bipartition of the terminals and the term *terminal cut* to refer to a cut which separates two arbitrary disjoint subsets of terminals (not necessarily a bipartition). With a slight abuse of notation, we refer to the value of the minimum cut for a bipartition/terminal cut as the value of the bipartition/terminal cut directly.

Before we present our results, we make a general remark about the property of all cut-preserving sparsifiers (and sketches) that is also used crucially in our lower bound proof. In [18], a *generalized* sparsifier is defined as a sparsifier that preserves the minimum cut between all disjoint subsets of terminals, i.e, terminal cuts and not only bipartition cuts. The authors then point out that their upper bound results, as well as the previous constructions of cut sparsifiers in [12], also satisfy this general definition. The following simple claim gives an explanation why all known cut sparsifiers satisfy this general definition.

▶ **Claim 1.** *Suppose H is a cut sparsifier of the graph $G(V,E)$ with terminals T that preserves the value of all bipartition cuts. Then, H also preserves the value of all terminal cuts.*

Proof. For any two disjoint subsets of terminals $A, B \subseteq T$, any cut separating A and B in G must form a bipartition $(S, \overline{S})$ of the terminals, and since H preserves the value of all minimum cuts like $(S, \overline{S})$, the (A,B) minimum cut value in H is also equal to the minimum cut value in G. In the case that H is a cut-preserving sketch, the (A,B) minimum cut can be answered by querying H with all bipartition cuts that separate (A,B), and outputting the smallest value. ◀

3.1 An $O(kC^2)$ Size Cut-Preserving Sketch

In this section, we construct a cut-preserving sketch for any digraph G and a set of terminals T. We achieve this by constructing an instance G' of the maximum matching problem in the dynamic sketching model and show that the value of any terminal cut (A,B) in G can be computed using a carefully designed query for the maximum matching size in G'. Our reduction is based on a classical result relating edge connectivity and bipartite matching due to [13] (see also [28], Section 16.7).

▶ **Theorem 7.** *For any directed graph G with a set of k terminals, there is an exact cut-preserving sketch that uses space $O(kC^2)$, where C is the total capacity of the edges incident on the terminals.*

Without loss of generality, we will replace each edge in G with capacity c_e with c_e parallel edges and still denote the new graph with G. Consider the following cut-preserving sketch.

A cut-preserving sketch

Input: A graph G with m edges and a set T of terminals.

- **Compression:** Construct a bipartite graph $G'(L, R, E')$ with terminals T' as follows and create a dynamic sketch for the maximum matching problem for G' and T'.
  - **a.** For each edge e in G, create two vertices e^- (in L) and e^+ (in R).
  - **b.** For any terminal q in T and any outgoing (resp. incoming) edge e of q, create a vertex $q^{\to e}$ in R (resp. $q^{\leftarrow e}$ in L). $q^{\to e}$ (resp. $q^{\leftarrow e}$), along with e^- (resp. e^+), belongs to the set T' of terminals in G'.
  - **c.** For each edge e in G, there is an edge between vertices e^- and e^+ in G'.
  - **d.** For any two edges e_1 and e_2 in G where the tail of e_1 is the head of e_2, there is an edge between the vertices e_1^+ and e_2^- in G'.
- **Extraction:** Given any two disjoint subsets $A, B \subseteq T$, let $Q_{A,B}$ be the query where for any terminal q in A (resp. in B) and any outgoing (resp. incoming) edge e of q, an edge between the vertices $q^{\to e}$ and e^- (resp. $q^{\leftarrow e}$ and e^+) is inserted in G'.
  Return $\nu(G'^{Q_{A,B}}) - m$, where $\nu(G'^{Q_{A,B}})$ is the maximum matching size in $G'^{Q_{A,B}}$.

The total number of terminals in G' is $2C$, and the total number of different A-B pairs (i.e., the total number of different possible queries) is at most 3^k. To ensure that every query is answered correctly, by Theorem 2, the sketch size is $O(kC^2)$. We now prove the correctness.

Proof. For any $A, B \subseteq T$, denote the value of the minimum A-B cut (which is equal to the edge-connectivity from A to B), by $c(A, B)$. Recall that for a graph G, $\nu(G)$ denotes the maximum matching size in G. We prove that $\nu(G'^{Q_{A,B}}) - m = c(A, B)$. Let M be the matching in $G^{Q_{A,B}}$ where for each edge e in G, e^- is matched with e^+; hence $|M| = m$.

We first show that if $c(A, B) = l$, then M can be augmented by l vertex disjoint paths and hence $\nu(G'^{Q_{A,B}}) \geq m + l$. There are l edge-disjoint path $P_1, P_2, \ldots, P_l$ from A to B in G. For each path $P_i = (e_1, e_2, \ldots, e_j)$, where e_1 starts with a terminal $q_a \in A$ and e_j ends with a terminal $q_b \in B$, create a path $P_i' = (q_a{}^{\to e_1}, e_1^-, e_1^+, e_2^-, \ldots, e_j^+, q_b{}^{\leftarrow e_j})$ in $G'^{Q_{A,B}}$. It is straightforward to verify that the P_i' paths are valid *vertex-disjoint* paths in $G'^{Q_{A,B}}$ and moreover, form disjoint augmenting paths of the matching M.

We now show that if the maximum matching M^* in G' is of size $m + l$, then $c(A, B) \geq l$. The symmetric difference between M and M^* forms a graph with l augmenting paths of the matching M. Each augmenting path must start and end with a vertex of the form $q^{\to e}$ or $q^{\leftarrow e}$ since they are the only vertices that are unmatched in M. Since every $q^{\to e}$ is in R and every $q^{\leftarrow e}$ is in L, each augmenting path must start with a $q^{\to e}$ vertex and ends with a $q^{\leftarrow e}$ vertex. Using the reversed transformation as in the previous case, the l augmenting paths can be converted into l edge disjoint paths from A to B in G. ◀

3.2 An $\tilde{\Omega}(C)$ Size Lower Bound

In this section, we provide a lower bound on the size of any cut-preserving sketch.

▶ **Theorem 8.** *For any integer $C > 0$, any cut-preserving sketch for k-terminal undirected graphs, where the total capacity of edges incident on the terminals is equal to C, requires $\Omega(C/\log C)$ bits.*

To prove this lower bound, we show how to encode a binary vector of length $N := \Omega(C/\log C)$ in an undirected graph G, so that given only a cut-preserving sketch for G, one can recover any entry of this vector. Standard information-theoretical arguments (similar to the lower bound for the MEMBERSHIP problem in Section 2) then imply that size of the cut-preserving sketch has to be of size $\Omega(N) = \Omega(C/\log C)$. We emphasize that while in the proof we assume the cut-preserving sketch has to return the value of minimum cuts between any subsets (A, B) of the terminals (even when they are not a bipartition), by Claim 1, this is without loss of generality; hence, the lower bound holds also for cut-preserving sketches that only guarantee to preserve minimum cuts for bipartitions.

Construction. Let $k' = k - 2$. For simplicity, assume k' is even, and let $N = \binom{k'}{\frac{k'}{2}}$. For any N-dimensional binary vector $\mathbf{v} \in \{0,1\}^N$, we define a graph $G_{\mathbf{v}}(V, E)$ as follows:

Vertices: The set of vertices of $G_{\mathbf{v}}$ is $V = \{s,t\} \cup \{q_1, \ldots, q_{k'}\} \cup \{u_1, \ldots, u_{k'}\} \cup \{v_1, \ldots, v_N\}$ and the k terminals are $T = \{s, q_1, \ldots, q_{k'}, t\}$.

Edges: Let $\mathcal{S} = \{S_1, \ldots, S_N\}$ be a collection of all $(k'/2)$-size subsets of $\{q_1, \ldots, q_{k'}\}$. The set of edges are defined as:

- For any $i \in [k']$, there is an edge (q_i, u_i) with capacity N.
- For any $i \in [k']$, there is an edge (s, u_i) with capacity N.
- For any $j \in [N]$, there is an edge (v_j, t) with capacity 1.
- A vertex u_j is connected to a vertex v_i with an edge of capacity 1 iff $\mathbf{v}_i = 1$ or $q_j \notin S_i$. Additionally, if u_j is connected to v_i, there are two more edges $f_1 = (s, v_i)$ and $f_2 = (u_j, t)$ each with capacity 1.
- There is an edge (s,t) with capacity $kN - m$, where m is the number of edges between $\{u_1, \ldots, u_{k'}\}$ and $\{v_1, \ldots, v_N\}$.

To recover the vector $\mathbf{v}$ from a cut-preserving sketch of $G_{\mathbf{v}}$, we will consider the terminal cuts (A, B) where $A = \{s\} \cup S_i$ for some $S_i \in \mathcal{S}$ and $B = \{t\}$. We further denote the terminal cut (A, B) corresponding to picking $S_i \in \mathcal{S}$ in the part A by $\text{TC}(S_i)$. We define the *output profile* of a graph $G_{\mathbf{v}} \in \mathcal{G}$ to be an N-dimensional vector $\mathbf{op}(G_{\mathbf{v}})$ where the i-th entry of $\mathbf{op}(G_{\mathbf{v}})$ is equal to the value of the terminal cut $\text{TC}(S_i)$. We show that there is a one-to-one correspondence between the vector $\mathbf{v}$ and $\mathbf{op}(G_{\mathbf{v}})$.

▶ **Lemma 9.** *Let $\mathbf{1}$ be the N-dimensional vector of all ones. There exists a value c independent of $\mathbf{v}$ such that $\mathbf{op}(G_{\mathbf{v}}) = \mathbf{v} + c \cdot \mathbf{1}$.*

Proof. Fix an $i \in [N]$ and consider $\text{TC}(S_i)$. We argue that the maximum flow value from $\{s\} \cup S_i$ to $\{t\}$ is $(k+1)N - 1 + \mathbf{v}_i$; the lemma then follows from the max-flow min-cut duality and the choice of $c = (k+1)N - 1$.

In $G_{\mathbf{v}}$, we can first send a flow of size kN from s to t by sending one unit of flow along every $s \to v_l \to u_j \to t$ path for any edge of the form (u_j, v_l) (m units of flow in total) and $kN - m$ units of flow over the (s,t) edge. After this process, the residual graph of $G_{\mathbf{v}}$ becomes

a directed graph where any edge of the form (u_j, v_l) is directed from u_j to v_l. Now consider any vertex v_p where $p \neq i$. There exists at least one terminal $q_j \in S_i$ (in fact, in $S_i \setminus S_p$), such that there is an edge between u_j and v_p in $G_{\mathbf{v}}$. Since in the residual graph of $G_{\mathbf{v}}$, this edge is directed from u_j to v_p, we can send one unit of flow over this edge also through the path $q_j \to u_j \to v_p \to t$. Hence, in $G_{\mathbf{v}}$, we can always send $kN + N - 1 = (k+1)N - 1$ units of flow from $\{s\} \cup S_i$ to $\{t\}$.

First suppose the i-th entry of $\mathbf{v}$ is equal to 1; then there is an edges from u_j to v_i for any $q_j \in S_i$. In particular, we can send one extra unit of flow over one of these edges to t, hence having a flow of size $(k+1)N$ entering t. Since the total capacity of the edges incident on t is $(k+1)N$, this ensures that the max-flow is also $(k+1)N$.

Now suppose the i-th entry of $\mathbf{v}$ is equal to 0. For the vertex v_i, by construction, there is no edge from any u_j to v_i, where $q_j \in S_i$. Hence in the residual graph of $G_{\mathbf{v}}$, there is no path from $\{s\} \cup S_i$ to $\{t\}$, meaning that the maximum flow in this case is $(k+1)N - 1$. This completes the proof. ◀

Proof of Theorem 8. Lemma 9 ensures that for any graph $G_{\mathbf{v}}$, there is a one-to-one correspondence between the value of i-th entry in $\mathbf{v}$ and i-th entry in $\mathbf{op}(G_{\mathbf{v}})$. Assuming that the cut-preserving sketch is able to answer each terminal cut (deterministically or even with a sufficiently small constant probability of error), we can recover i-th bit of $\mathbf{v}_i$, from the i-th index in $\mathbf{op}(G_{\mathbf{v}})$ with a constant probability. Standard information-theoretical arguments imply that the size of the cut-preserving has to be $\Omega(N)$. Moreover, since $N = 2^{\Omega(k)} = \Omega(C/k) = \Omega(C/\log C)$ in this construction, we obtain the final bound of $\Omega(C/\log C)$ bits on the sketch size. ◀

We should point out for the case of randomized cut-preserving sketches that are only guaranteed to have a constant probability of failure over bipartition cuts (and not necessarily terminal cuts), we first need to reduce the probability of error to 2^{-k} before performing the described construction (and applying Claim 1) which results in a lower bound of $\Omega(C/\log^2 C)$.

We further point out that, as a corollary of Theorem 8, we also obtain a simple proof for a lower bound of $2^{\Omega(k)}$ on size of cut-preserving sketches (see [18, 15]).

4 The s-t Edge-Connectivity Problem

In this section, we study dynamic sketching for the s-t edge-connectivity problem. As it turns out, any cut-preserving sketch can be directly adapted to a dynamic sketching scheme for the s-t edge-connectivity problem as follows. Given a graph G with a set T of k terminals and two designated vertices s and t, create a cut-preserving sketch for G with terminals $T \cup \{s, t\}$. Note that given a query Q (i.e., a set of edges among T), the s-t minimum cut (which is equal to the s-t edge-connectivity) will partition $T \cup \{s, t\}$ into two sets T_s and T_t, where T_s contains s and T_t contains t. Hence, the minimum cut from T_s to T_t is equal to the minimum cut from s to t. The cut-preserving sketch can answer the minimum cut from T_s to T_t in the original graph, and the additional cut value caused by the query is simply the total number of the edges from T_s to T_t. Therefore, if we enumerate all possible partitions of the terminals that separate s and t, and compute the minimum cut for each partition as above, the smallest minimum cut among those partitions is equal to the minimum cut from s to t.

Nevertheless, by our lower bound on the size of cut-preserving sketches (Theorem 8), a dynamic sketching scheme constructed as above, will have a linear dependency on the total degree of the vertices in $T \cup \{s, t\}$, which could be as large as the number of vertices in the graph. To resolve this issue, we propose a scheme which directly uses our dynamic

sketching scheme for the maximum matching problem and achieve a sketch of size $O(k^4)$. The reduction is in the same spirit as the one we used for cut-preserving sketches. But note that the main differences is that unlike the case for cut-preserving sketches, in the dynamic sketching problem the set of edges in the original graph changes which require additional care.

▶ **Theorem 10.** *For any $\delta > 0$, there exists a randomized k-dynamic sketching scheme for the s-t edge-connectivity problem with a sketch of size $O(k^4 \log(1/\delta))$, which answers any query correctly with probability at least $1 - \delta$.*

Given a digraph G with two designated vertices s and t, along with a set of T terminals, recall that, for the query $Q_\forall$ where an edge is inserted between each (ordered) pair of terminals, $G^{Q_\forall}$ denotes the graph after applying the query $Q_\forall$ to G. Assume s does not have any incoming edge and t does not have any outgoing edge, since removing them will not affect the s-t edge-connectivity. We further assume that s are t are not terminals. This is without loss of generality since we can create two vertices s' and t' that are not terminals, while adding $d^+(s)$ (resp. $d^-(t)$) new vertices and for each of these vertices v, adding an edge from s' to v and v to s (resp. t to v and v to t'). In this new graph, the s'-t' edge connectivity is equal to the s-t edge-connectivity in G and s', t' are not terminals. Consider the following dynamic sketching scheme.

A dynamic sketching scheme for the s-t edge-connectivity problem

Input: A graph G with m edges, two designated vertices s and t, and a set T of k terminals.

- **Compression:** Construct a bipartite graph $G'(L, R, E')$ with a set T' of terminals as follows and create a dynamic sketch for the maximum matching problem for G' and T'.
  - **a.** For each edge e in $G^{Q_\forall}$, if e starts with s, create a vertex e^+ (in R), if e ends with t create a vertex e^- (in L), otherwise, create two vertices e^+ (in R) and e^- (in L).
  - **b.** For each edge e between two terminals in G, create two vertices $\hat{e}^+$ and $\hat{e}^-$; $\hat{e}^-$ and $\hat{e}^+$, along with e^- and e^+, are terminals of G'.
  - **c.** For each edge e in G where both e^- and e^+ exist, there is an edge between e^- and e^+.
  - **d.** For any two edges e_1 and e_2 in $G^{Q_\forall}$ where the tail of e_1 is the head of e_2, there is an edge between e_1^+ and e_2^-.
- **Extraction:** Given any query Q of G, let Q' be the query of G' where
  - **a.** For each edge e in Q, insert an edge between e^- and e^+.
  - **b.** For each edge e in $Q_\forall \setminus Q$, insert an edge between e^- and $\hat{e}^+$, and between e^+ and $\hat{e}^-$.
  - **c.** Let the maximum matching size of $G'^{Q'}$ be ν. Output $\nu - (m + 2k^2 - |Q|)$.

The total number of terminals in G' is $4k^2$. Hence by Theorem 2, the sketch size is $O(k^4 \log(1/\delta))$. The correctness of the reduction is similar in spirit to the proof of Theorem 7 and is deferred to the full version [5] (see Section 4.1).

We conclude this section by remarking that there exists an equivalence between dynamic sketching the capacitated version of the s-t edge connectivity problem (i.e., the s-t maximum flow problem) and cut-preserving sketches. In particular,

▶ **Theorem 11.** *Any cut-preserving sketch can be adapted to a dynamic sketching scheme for the s-t maximum flow problem while increasing the number of terminals by at most* 2*, and vice versa.*

The proof of this theorem together with a detailed discussion on the similarity of the *s-t* maximum flow problem and cut-preserving sketches is provided in Section 4.2 of the full version of the paper [5]. However, we point out here that Theorem 11 combined with Theorem 8, proves a similar $2^{\Omega(k)}$ lower bound on size of dynamic sketches for the *s-t* maximum flow problem. In other words, this problem does not admit a compact dynamic sketch.

5 Conclusions

In this paper we have introduced *dynamic sketching*, a new approach for compressing data sets separated into static and dynamic parts. We studied dynamic sketching for graph problems where the dynamic part consists of k vertices and the edges between them may get modified in an arbitrary manner (a query). We showed that the maximum matching problem admits a sketch of size $O(k^2)$ and the space bound is tight. Moreover, this sketch can be used to obtain cut-preserving sketches of size $O(kC^2)$, and dynamic sketches for *s-t* edge-connectivity of size $O(k^4)$.

There are problems (even in P) for which any dynamic sketch requires $2^{\Omega(k)}$ space. An interesting direction for future work is to identify broad classes of problems that admit compact dynamic sketches, i.e, sketches of size poly(k).

Some data compression schemes (most notably, cut sparsifiers and kernelization results) generate as compressed representation an instance of the original problem, while the sketches we introduced do not fall into this category. A natural question is to understand if there exist polynomial-size "sparsifier-like" compressed representations for matchings and *s-t* edge connectivity in the dynamic sketching model.

Finally, while our work narrows the gap between upper and lower bounds on the size of a cut-preserving sketches, it remains an intriguing open question to get an asymptotically tight bound on the size of cut-preserving sketches.

Acknowledgments. We are grateful to Chandra Chekuri and Michael Saks for helpful discussions.

References

1 Farid M. Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 157(2):139–159, 1996.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 459–467, 2012.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 5–14, 2012.

4 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\varepsilon)$-approximate flow sparsifiers. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 279–293, 2014.

5 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Dynamic sketching for graph optimization problems with applications to cut-preserving sketches. *CoRR*, abs/1510.03252, 2015.

6 Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 265–274, 2010.

7 Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 673–688, 2012.

8 Daniel Deutch, Zachary G. Ives, Tova Milo, and Val Tannen. Caravan: Provisioning for what-if analysis. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR*, 2013.

9 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX, and 14th International Workshop, RANDOM*, pages 152–165, 2010.

10 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *Automata, Languages and Programming: 31st International Colloquium, ICALP*, pages 531–543, 2004.

11 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC*, pages 133–142, 2008.

12 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998.

13 Alan J Hoffman. Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. In *Proc. Sympos. Appl. Math*, volume 10, pages 113–127. World Scientific, 1960.

14 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 561–570, 2014.

15 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014.

16 Stefan Kratsch and Magnus Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 94–103, 2012.

17 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 450–459, 2012.

18 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1789–1799, 2013.

19 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 47–56, 2010.

20 L. Lovász and D. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. American Mathematical Soc., 2009.

21 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

22 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009.

23 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

24 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 3–12, 2009.

25 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

26 Michael O. Rabin and Vijay V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, 1989.

27 Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011.

28 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

29 William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

Weighted Strategy Logic with Boolean Goals Over One-Counter Games*

Patricia Bouyer, Patrick Gardy, and Nicolas Markey

LSV – CNRS, ENS Cachan, Université Paris-Saclay, France
firstname.lastname@lsv.fr

Abstract

Strategy Logic is a powerful specification language for expressing non-zero-sum properties of multi-player games. SL conveniently extends the logic ATL with explicit quantification and assignment of strategies. In this paper, we consider games over one-counter automata, and a quantitative extension 1cSL of SL with assertions over the value of the counter. We prove two results: we first show that, if decidable, model checking the so-called *Boolean-goal* fragment of 1cSL has non-elementary complexity; we actually prove the result for the Boolean-goal fragment of SL over finite-state games, which was an open question in [32]. As a first step towards proving decidability, we then show that the Boolean-goal fragment of 1cSL over one-counter games enjoys a nice periodicity property.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Temporal logics, multi-player games, strategy logic, quantitative games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.69

1 Introduction

Model checking. Model checking [19] has been developed for almost 40 years as a formal method for verifying correctness of computerized systems: this technique first consists in representing the system under study as a mathematical model (a finite-state transition system (a.k.a. Kripke structure), in the most basic setting), expressing the correctness property in some logical formalism (usually, using various *temporal logics* such as LTL [35] or CTL [18, 36]), and running an algorithm that exhaustively explores the set of behaviours of the model for proving or disproving the property.

Over the years, model checking has been extended in various directions, in order to take into account richer models and more precise properties. Several families of quantitative models (e.g. weighted Kripke structures [12], counter automata [25], timed automata [1]) and temporal logics [29, 24, 2, 7, 9, among others] have been defined and studied. Those formalisms conveniently extend the qualitative setting; they provide powerful ways of representing quantities, while in several cases keeping reasonably efficient model-checking algorithms.

Multi-agent systems (a.k.a. graph games [42, 4]) form another direction where model checking has been extended for reasoning about the interactions between components of a computerized system. Temporal logics have been extended accordingly [3, 16, 34, 20], in order to express the existence of *winning strategies* in multi-player games. Among the most popular approaches, the logic ATL [3] has limited expressive power but enjoys reasonably efficient model-checking algorithms, while the more expressive Strategy Logic (SL) [16, 34] extends LTL with explicit manipulation of strategies, and can express very rich non-zero-sum

* This work was partially supported by FP7 projects Cassting (601148) and ERC EQualIS (308087).

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 69–83

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

properties of games, including equilibria; however, model checking SL is non-elementary. Several fragments of SL have recently been introduced in order to mitigate the complexity of the model-checking problem while retaining the interesting aspects of SL [33, 13].

Quantitative games, combining both extensions, have also been widely considered. This includes games on weighted graphs [23, 14, 31, 8], games on counter systems or VASS [39, 11], or timed games [5, 21]. A large part of these works have focused on "simple" objectives, such as mean-payoff objectives [23], energy constraints [14, 8], or combinations thereof [15, 26].

Our contribution. In this paper, we consider a quantitative extension of SL over quantitative games. While such extensions have already been proven decidable for ATL [31, 43], we focus here on a quantitative extension of the richer logic SL, more specifically, its so-called *Boolean-goal* fragment SL[BG] [32]. SL with Boolean goals restricts SL by preventing arbitrary nesting of strategy quantifiers within temporal modalities. This and several other fragments of SL have been introduced in [32] with the aim of getting more efficient model-checking algorithms. However, while several fragments have been shown to have *2*-EXPTIME model-checking algorithms, the exact complexity of SL[BG] remained open.

We prove that model checking (the flat fragment of) SL[BG] is Tower-complete, thus negatively answering the open question whether SL[BG] would enjoy more efficient model-checking algorithm than SL. This hardness result obviously extends to the quantitative version 1cSL[BG] of SL[BG] over one-counter games. On the way to proving decidability of the model-checking problem for this logic, we then show that 1cSL[BG] over one-counter games enjoys a nice periodicity property: for any given formula, there is a threshold above which truth value of the formula is periodic (w.r.t. the value of the counter).

Related works. Several works have focused on one-counter models: two-player games with parity objectives have been proven PSPACE-complete [39]; this was recently extended to a quantitative extension of ATL [43], which is thus closely related to our paper. Model checking LTL and CTL over one-counter automata is also PSPACE-complete [28, 27]. Quantitative extensions of those logics have been studied in [22, 7, 9]. In many cases, they lead to undecidability of the model-checking problem. Games on VASS have also been considered, but reachability is only decidable in restricted cases [11, 37].

Games over integer-weighted graphs have a different flavour, as the behaviours do not depend on the value of the accumulated weight. Those games have been extensively considered with various quantitative objectives (e.g. mean-payoff [23, 44], energy [14, 8], and combinations thereof [15, 17]), and with objectives expressed in temporal logics [31, 6].

2 Definitions

▶ **Definition 1.** Let AP be a set of atomic propositions, and Agt be a set of agents. A *1-counter concurrent game structure* (1cCGS for short) is a tuple $\mathcal{G} = \langle \mathsf{Loc}, \mathsf{label}, \mathsf{Act}, \mathsf{Tab}_{\{0,1\}}, \mathsf{Wgt}_{\{0,1\}} \rangle$ where

- Loc is a finite set of locations;
- $\mathsf{label} \colon \mathsf{Loc} \to 2^{\mathsf{AP}}$ labels locations with atomic propositions;
- Act is a finite set of actions;
- $\mathsf{Tab}_0 \colon \mathsf{Loc} \times \mathsf{Act}^{\mathsf{Agt}} \to \mathsf{Loc}$ and $\mathsf{Tab}_1 \colon \mathsf{Loc} \times \mathsf{Act}^{\mathsf{Agt}} \to \mathsf{Loc}$ are two transition tables;
- $\mathsf{Wgt}_0 \colon \mathsf{Loc} \times \mathsf{Act}^{\mathsf{Agt}} \to \{0, 1\}$ and $\mathsf{Wgt}_1 \colon \mathsf{Loc} \times \mathsf{Act}^{\mathsf{Agt}} \to \{-1, 0, 1\}$ assign a weight to each transition of the transition tables.

A *finite path* in a 1cCGS $\mathcal{G}$ is a finite non-empty sequence of configurations $\rho = \gamma_0\gamma_1\gamma_2\dots\gamma_k$, where for all $0 \leq i \leq k$, the configuration γ_i is a pair (ℓ_i, c_i) with $\ell_i \in \mathsf{Loc}$ and $c_i \in \mathbb{N}$. For such a path, we denote by $\mathsf{last}(\rho)$ its last element γ_k, and we let $|\rho| = k$. number of transitions An *infinite path* is an infinite sequence of configurations with the same property. We denote by Path (resp. $\mathsf{InfPath}$) the set of finite (resp. infinite) paths. The length of an infinite path is $+\infty$. For $0 \leq i < |\rho|$, $\rho(i)$ represents the $i+1$-th element γ_i of ρ. For a path ρ and $0 \leq i < |\rho|$, we denote by $\rho_{\leq i}$ the prefix of ρ until position i, i.e. the finite path $\rho(0)\rho(1)\dots\rho(i)$.

A *strategy* for some agent $a \in \mathsf{Agt}$ is a function $\sigma_a \colon \mathsf{Path} \to \mathsf{Act}$. We write Strat for the set of strategies. Given a finite path (or *history*) in the game, a strategy σ_a returns the action that agent a will play next. A strategy σ_A for a coalition of agents $A \subseteq \mathsf{Agt}$ is a function assigning a strategy $\sigma_A(a)$ to each agent $a \in A$. Given a strategy σ_A for coalition A, we say that a path ρ respects σ_A after a finite prefix π if, writing $\rho(i) = (\ell_i, c_i)$ for all $0 \leq i \leq |\rho|$, the following two conditions hold:

- for all $0 \leq i < |\pi|$, we have $\rho(i) = \pi(i)$
- for all $|\pi| \leq i < |\rho| - 1$, we have that $\ell_{i+1} = \mathsf{Tab}_s(\ell_i, \mathbf{m})$ and $c_{i+1} = c_i + \mathsf{Wgt}_s(\rho_{\leq i}, \mathbf{m})$, where $s = 0$ if $c_i = 0$ and $s = 1$ otherwise, and $\mathbf{m}$ is an action vector satisfying $\mathbf{m}(a) = \sigma_A(a)(\rho_{\leq i})$ for all $a \in A$.

Notice that the value of the counter always remains nonnegative as Wgt_0 only returns nonnegative values. Given a finite path π, we denote by $\mathsf{Out}(\pi, \sigma_A)$ the set of paths that respect the strategy σ_A after prefix π. Notice that if σ_A assigns a strategy to all the agents, then $\mathsf{Out}(\pi, \sigma_A)$ contains a single path, which we write $\mathsf{out}(\pi, \sigma_A)$.

▶ Remark. Several semantics have been given to quantitative games, see [37]. The semantics chosen here, with zero tests (using $\mathsf{Tab}_0, \mathsf{Tab}_1$), allows to easily express the three semantics studied in [37]. Hence our algorithms apply in all these settings. It is worth noticing that the hardness proof holds for the non-quantitative setting, hence also for all three semantics mentioned above.

We now define our weighted extension of Strategy Logic [16, 34]:

▶ **Definition 2.** Let AP be a set of atomic propositions, Agt be a set of agents, and Var be a finite set of strategy variables. Formulas in 1cSL are built from the following grammar:

$$\text{1cSL} \ni \phi ::= p \mid \mathsf{cnt} \in S \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{X}\,\phi \mid \phi\,\mathbf{U}\,\phi \mid \exists x.\ \phi \mid \mathsf{bind}(a \mapsto x).\ \phi$$

where p ranges over AP, S is a subset of $\mathbb{N}$ that can be described as $S^1_{\mathsf{fin}} \cup \left(S^2_{\mathsf{fin}} + k \cdot \mathbb{N}\right)$, where S^i_{fin} are finite subsets of $\mathbb{N}$ and $k \in \mathbb{N}$ is a period[1], x ranges over Var, and a ranges over Agt. The logic SL is the fragment of 1cSL where no counter constraint $\mathsf{cnt} \in S$ or $\mathsf{cnt} \in S_{[k]}$ is used. The logic 1cLTL is the fragment of 1cSL where no strategy quantifiers $\exists x.\ \phi$ and no strategy bindings $\mathsf{bind}(a \mapsto x).\ \phi$ are used. Finally, LTL is the intersection of SL and 1cLTL.

The set of *free agents and variables* of a formula ϕ of 1cSL, which we write $\mathsf{free}(\phi)$, contains the agents and variables that have to be associated with a strategy before ϕ can be

[1] This allows to express standard counter constraints like $\mathsf{cnt} \geq 5$ (using negation) or periodic constraint like $\mathsf{cnt} = 4 \bmod 7$. Notice that our periodicity result is not a consequence of the periodicity of the quantitative assertions, and would also hold with assertions of the form $\mathsf{cnt} \sim n$.

evaluated. It is defined inductively as follows:

$$\mathsf{free}(p) = \varnothing \quad \text{for } p \in \mathsf{AP} \qquad \mathsf{free}(\mathbf{X}\,\phi) = \mathsf{Agt} \cup \mathsf{free}(\phi)$$
$$\mathsf{free}(\mathsf{cnt} \in S) = \varnothing \quad \text{for } n \in \mathbb{N} \qquad \mathsf{free}(\phi\,\mathbf{U}\,\psi) = \mathsf{Agt} \cup \mathsf{free}(\phi) \cup \mathsf{free}(\psi)$$
$$\mathsf{free}(\neg\,\phi) = \mathsf{free}(\phi) \qquad \mathsf{free}(\phi \vee \psi) = \mathsf{free}(\phi) \cup \mathsf{free}(\psi)$$
$$\mathsf{free}(\exists x.\ \phi) = \mathsf{free}(\phi) \setminus \{x\} \quad \mathsf{free}(\mathsf{bind}(a \mapsto x).\ \phi) = \begin{cases} \mathsf{free}(\phi) & \text{if } a \notin \mathsf{free}(\phi) \\ (\mathsf{free}(\phi) \cup \{x\}) \setminus \{a\} & \text{otherwise} \end{cases}$$

A formula ϕ is *closed* if $\mathsf{free}(\phi) = \emptyset$.

We can now define the semantics of 1cSL. Let $\mathcal{G}$ be a 1cCGS, π be a path, i be a position along π, and $\chi \colon \mathsf{Var} \cup \mathsf{Agt} \dashrightarrow \mathsf{Strat}$ be a partial valuation (or context) with domain $\mathsf{dom}(\chi)$. Let $\phi \in \mathsf{SL}$ such that $\mathsf{free}(\phi) \subseteq \mathsf{dom}(\chi)$. Whether ϕ holds true at position i along π within context χ is defined inductively as follows:

$$\begin{array}{lll}
\mathcal{G}, \pi, i \models_\chi p & \text{iff} & p \in \mathsf{label}(\ell_i) \quad (\text{writing } \pi(i) = (\ell_i, c_i)) \\
\mathcal{G}, \pi, i \models_\chi \mathsf{cnt} \in S & \text{iff} & c_i \in S \quad (\text{writing } \pi(i) = (\ell_i, c_i)) \\
\mathcal{G}, \pi, i \models_\chi \neg\,\phi_1 & \text{iff} & \mathcal{G}, \pi, i \not\models_\chi \phi_1 \\
\mathcal{G}, \pi, i \models_\chi \phi_1 \vee \phi_2 & \text{iff} & \mathcal{G}, \pi, i \models_\chi \phi_1 \text{ or } \mathcal{G}, \pi, i \models_\chi \phi_2 \\
\mathcal{G}, \pi, i \models_\chi \mathbf{X}\,\phi_1 & \text{iff} & \mathcal{G}, \rho, i+1 \models_\chi \phi_1 \quad (\text{writing } \rho = \mathsf{out}(\pi_{\leq i}, \chi_{|\mathsf{Agt}})) \\
\mathcal{G}, \pi, i \models_\chi \phi_1\,\mathbf{U}\,\phi_2 & \text{iff} & \exists k \geq i.\ \mathcal{G}, \rho, k \models_\chi \phi_2 \text{ and} \\
 & & \forall i \leq j < k.\ \mathcal{G}, \rho, j \models_\chi \phi_1 \quad (\text{writing } \rho = \mathsf{out}(\pi_{\leq i}, \chi_{|\mathsf{Agt}})) \\
\mathcal{G}, \pi, i \models_\chi \exists x.\ \phi_1 & \text{iff} & \exists \sigma \in \mathsf{Strat}.\ \mathcal{G}, \pi, i \models_{\chi[x \mapsto \sigma]} \phi_1 \\
\mathcal{G}, \pi, i \models_\chi \mathsf{bind}(a \mapsto x).\ \phi_1 & \text{iff} & \mathcal{G}, \pi, i \models_{\chi[a \mapsto \chi(x)]} \phi_1
\end{array}$$

Notice that the constraint that $\mathsf{free}(\phi) \subseteq \mathsf{dom}(\chi)$ is preserved at each step.

▶ Remark. One may notice that the relation $\mathcal{G}, \pi, i \models_\chi \phi$ does not depend on the suffix of π after position i. Moreover, writing $\sigma_{\overrightarrow{\pi_{\leq i}}}$ for the strategy σ' such that $\sigma'(\rho) = \sigma(\pi_{\leq i} \cdot \rho)$, it is easily proved that $\mathcal{G}, \pi, i \models_\chi \phi$ if, and only if, $\mathcal{G}, \pi', 0 \models_{\chi'} \phi$, where $\chi'(x) = \chi(x)_{\overrightarrow{\pi_{\leq i}}}$ for all $x \in \mathsf{Var} \cup \mathsf{Agt}$ (we will later write $\chi_{\overrightarrow{\pi_{\leq i}}}$ for χ'). As the satisfaction relation does not depend on the suffix of π after position i, we may also write $\mathcal{G}, \gamma \models_{\chi'} \phi$, where $\gamma = \pi(i)$. In the sequel, we may even omit to mention $\mathcal{G}$ when it is clear from the context, and simply write $\gamma \models_\chi \phi$.

▶ Remark. We write $\langle\!\langle a \rangle\!\rangle\,\phi$ as a shorthand for $\exists \sigma_a.\ \mathsf{bind}(a \mapsto \sigma_a).\ \phi$, when we do not need to have hands on σ_a in the rest of the formula. Similarly, $[\![a]\!]\,\phi$ stands for $\neg\,\langle\!\langle a \rangle\!\rangle\,\neg\,\phi$. This construct $\langle\!\langle a \rangle\!\rangle\,\phi$ precisely corresponds to the strategy quantification used in the logic ATL_{sc} [30], but it should be noticed that it does *not* correspond to the strategy quantifier of ATL [3].

In the sequel, we also use other classical shorthands such as $\top$, defined as $p \vee \neg\,p$ for some p (hence it is always true); $\mathbf{F}\,\phi$ as a shorthand for $\top\,\mathbf{U}\,\phi$, meaning that ϕ holds at a later position; and $\mathbf{G}\,\phi$, defined as $\neg\,\mathbf{F}\,\neg\,\phi$, meaning that ϕ holds true at every future position.

Several fragments of SL have recently been defined and studied [32]. Those fragments restrict the use of strategy bindings and quantifications. In the present paper, we are mainly interested in the quantitative extension of the fragment SL[BG]. Before defining 1cSL[BG], we first introduce its *flat* fragment $\mathsf{1cSL}^0[\mathsf{BG}]$:

$$\begin{aligned}
\mathsf{1cSL}^0[\mathsf{BG}] \ni \phi &::= \neg\,\phi \mid \phi \vee \phi \mid \exists x.\ \phi \mid \mathsf{bind}(a \mapsto x).\ \phi \mid \psi \\
\psi &::= p \mid \mathsf{cnt} \in S \mid \neg\,\psi \mid \psi \vee \psi \mid \mathbf{X}\,\psi \mid \psi\,\mathbf{U}\,\psi
\end{aligned}$$

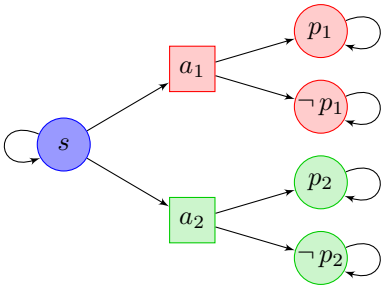


Figure 1 The 3-player turn-based game for the reduction to SL model checking.

▶ Remark. Any closed formula φ in $\mathsf{1cSL}^0[\mathsf{BG}]$ can be written in *prenex form* as

$$\wp(\mathsf{Var}).\ f\Big((\beta_i(\mathsf{Agt},\mathsf{Var}).\ \psi_i)_{1\le i\le n}\Big)$$

where $\wp(\mathsf{Var})$ is a series of strategy quantifiers involving all variables in Var, f is a Boolean combination over n atoms, and for every $1 \le i \le n$, β_i assigns a strategy from Var to each agent of Agt, and ψ_i is a 1cLTL formula.

1cSL[BG] then extends $\mathsf{1cSL}^0[\mathsf{BG}]$ by allowing nesting *closed* formulas at the level of atomic propositions. Formally, we defined the depth-i fragment as

$$\begin{aligned}\mathsf{1cSL}^i[\mathsf{BG}] \ni \phi &::= \neg\phi \mid \phi\vee\phi \mid \exists x.\ \phi \mid \mathsf{bind}(a\mapsto x).\ \phi \mid \psi\\ \psi &::= p \mid \phi_{i-1} \mid \mathsf{cnt}\in S \mid \neg\psi \mid \psi\vee\psi \mid \mathbf{X}\,\psi \mid \psi\,\mathbf{U}\,\psi\end{aligned}$$

where ϕ_{i-1} ranges over closed formulas of $\mathsf{1cSL}^{i-1}[\mathsf{BG}]$. We let 1cSL[BG] be the union of the fragments $\mathsf{1cSL}^i[\mathsf{BG}]$ for all $i \in \mathbb{N}$. It can be checked that if we drop the quantitative constraints from 1cSL[BG], we precisely get the logic SL[BG] of [32].

3 Hardness of SL[BG] model checking

In this section, we prove that the model-checking problem for SL[BG] is Tower-hard (the complexity class Tower is the union of all classes k-EXPTIME when k ranges over $\mathbb{N}$ [38]). We actually prove the result for (the flat fragment of) SL[BG], closing a question left open in [32].

▶ **Theorem 3.** *Model checking SL[BG], and hence 1cSL[BG], is* Tower*-hard.*

We give a sketch of the proof here, and develop the full proof in [10].

Sketch of proof. We prove this result by encoding the satisfiability problem for QLTL into the model-checking problem for SL[BG]. QLTL is the extension of LTL with quantification over atomic propositions [40]: formulas in QLTL are of the form $\Phi = \forall p_1 \exists p_2 \ldots \forall p_{n-1} \exists p_n.\ \varphi$ where ϕ is in LTL. Notice that we only consider strictly alternating formulas for the sake of readability. The general case can be handled similarly. Formula $\exists p.\ \varphi$ holds true over a word $w\colon \mathbb{N} \to 2^{\mathsf{AP}}$ if there exists a word $w'\colon \mathbb{N} \to 2^{\mathsf{AP}}$ with $w'(i)\cap(\mathsf{AP}\setminus\{p\}) = w(i)\cap(\mathsf{AP}\setminus\{p\})$ and $w' \models \varphi$ for all i. Universal quantification is defined similarly. It is well-known that model checking (and satisfiability) of QLTL is Tower-complete [41]. We reduce the satisfiability of QLTL into a model-checking problem for a SL[BG] formula involving $n+4$ players (where n is the number of quantifiers in the QLTL formula), and three additional quantifier alternations.

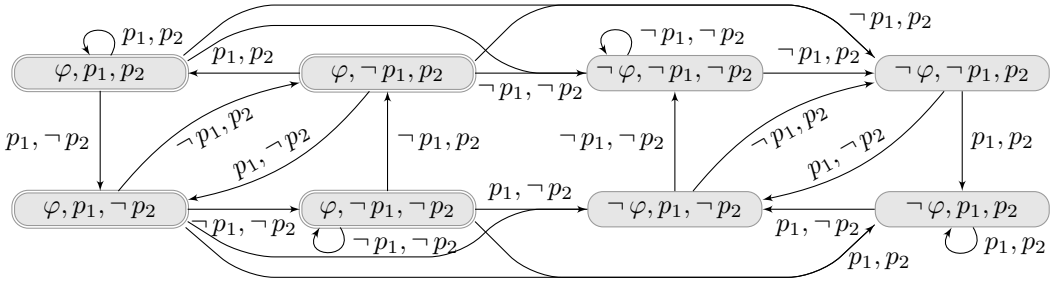


Figure 2 Büchi automaton for $\mathbf{G}\,(p_2 \Leftrightarrow \mathbf{X}\,p_1)$.

Before developing this technical encoding, we first present an example of a reduction to plain SL, which already contains most of the intuitions of our reduction to SL[BG]. Consider the QLTL formula

$$\Phi = \forall p_1.\ \exists p_2.\ \mathbf{G}\,(p_2 \Leftrightarrow \mathbf{X}\,p_1).$$

To solve the satisfiability problem of this formula via SL, we use the three-player turn-based game depicted on Fig. 1. In that game, Player Blue controls the blue state s, while Players Red and Green control the square states a_1 and a_2, respectively. Fix a strategy of Player Red: this strategy will be evaluated only in red state a_1, hence after histories of the form $s^n \cdot a_1$. Hence a strategy of Player Red can be seen as associating with each integer n a value for p_1. In other words, a strategy for Player Red defines a labeling of the time line with atomic proposition p_1. Similarly for Player Green and proposition p_2.

It remains to use this correspondence for encoding our QLTL formula. We have to express that for any strategy σ_{Red} of Player Red, there is a strategy σ_{Green} of Player Green under which, at each step along the path that stays in s forever, Player Blue can enforce $\mathbf{X}\,\mathbf{X}\,p_2$ if, and only if, he can enforce $\mathbf{X}\,\mathbf{X}\,p_1$ one step later. In the end, the formula reads as follows:

$$[\cdot\mathsf{Red}\cdot]\ \langle\cdot\mathsf{Green}\cdot\rangle\ \langle\cdot\mathsf{Blue}\cdot\rangle\ \mathbf{G}\left(s \wedge (\ \langle\cdot\mathsf{Blue}\cdot\rangle\ \mathbf{X}\,\mathbf{X}\ p_2\ ) \Leftrightarrow (\mathbf{X}\ \langle\cdot\mathsf{Blue}\cdot\rangle\ \mathbf{X}\,\mathbf{X}\ p_1\ )\right) \tag{1}$$

One may notice that the above property is not in SL[BG]: for instance, the subformula $\langle\cdot\mathsf{Blue}\cdot\rangle\ \mathbf{X}\,\mathbf{X}\ p_2$ is not closed. We provide a different construction, refining the ideas above, in order to reduce QLTL satisfiability to SL[BG] model checking.

In order to do so, we take another approach for encoding the LTL formula, since our technique of encoding p_i with $\langle\cdot\mathsf{Blue}\cdot\rangle\ \mathbf{X}\,\mathbf{X}\ p_i$ is not compatible with getting a formula in SL[BG]. Instead, we will use a Büchi automaton encoding the formula; another player, say Player Black, will be in charge of selecting states of the Büchi automaton at each step. Using the same trick as above in the game structure on the left of Fig. 3, a strategy for Player Black can be seen as a mapping from $\mathbb{N}$ to states of the Büchi automaton. Our formula will ensure that this sequence of states is in accordance with the atomic propositions selected by the square players in states a_i, and that it forms an accepting run of the Büchi automaton.

For our example, an eight-state Büchi automaton associated with the (LTL part of the) QLTL formula is depicted on Fig. 2. Notice that smaller automata exist for this property (for instance, the four states on the right could be merged into a single one), but for technical reasons in our construction, we require that each state of the Büchi automaton corresponds to a single valuation of the atomic propositions, hence the number of states must be a multiple of $2^{|\mathsf{AP}|}$. Accordingly, we augment our game structure of Fig. 1 with eight extra states, as depicted on the left of Fig. 3. Again, a strategy of Player Black (controlling state b) defines a sequence of states of the Büchi automaton.

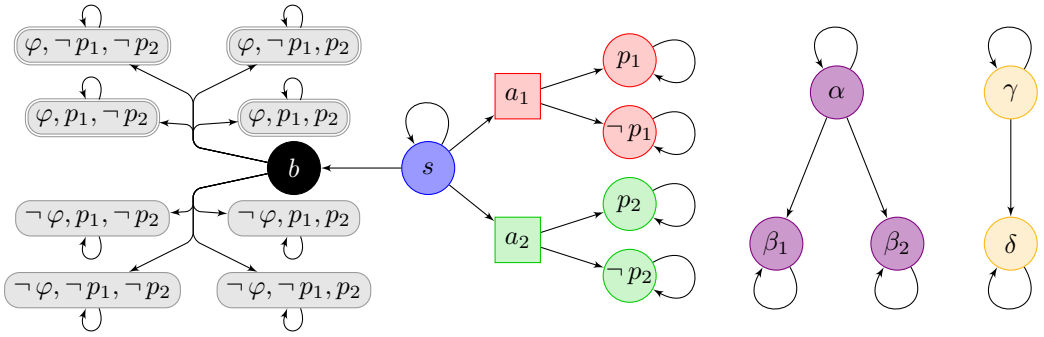


Figure 3 The concurrent game for the reduction to SL[BG] model checking.

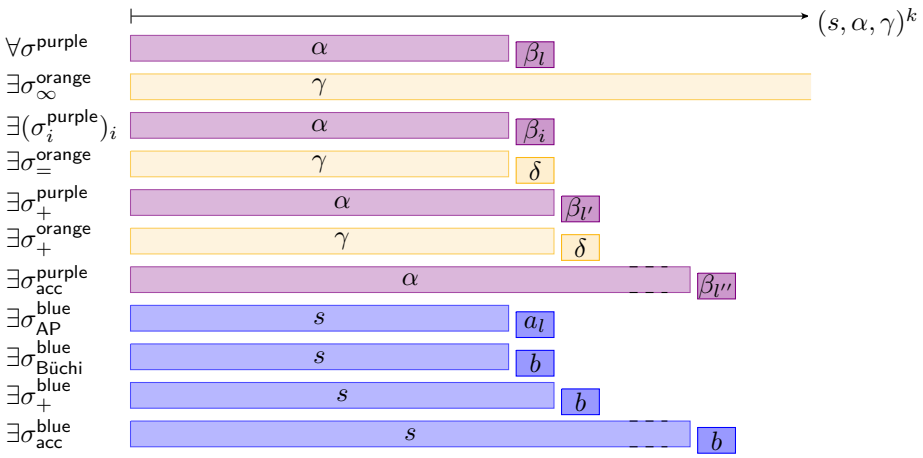


Figure 4 Visualization of the strategies selected by Ψ_{aux} on history $(s, \alpha, \gamma)^k$.

It then remains to "synchronize" the run of the Büchi automaton with the valuations of the atomic propositions, selected by the players controlling the square states. This is achieved by taking the product of the game we just built with two extra one-player structures, as depicted on the right of Fig. 3. The product gives rise to a concurrent game, where one transition is taken simultaneously in the main structure and in the Purple and Orange structures. In this product, as long as Player Blue remains in s and Player Purple remains in α, a strategy of Player Orange (controlling state γ) either remains in γ forever, or it can be characterized by a value $n \in \mathbb{N}$. Similarly, as long as Player Blue remains in s and Player Orange remains in γ, a strategy of Player Purple (controlling state α) either loops forever in α, or can be uniquely characterized by a pair (k, p_l), where k is the number of times the loop over α is taken before entering state β_l corresponding to $p_l \in \mathsf{AP}$.

Our construction can then be divided in two steps:

- First, with any strategy of Player Purple (characterized by (k, p_l) for the interesting cases), we associate auxiliary strategies of Players Blue, Purple and Orange satisfying certain properties, that can be enforced by an SL[BG] formula Ψ_{aux}; Fig. 4 should help visualizing the associated strategies; in particular, strategies $\sigma^{\text{orange}}_{+}$, σ^{blue}_{+} and $\sigma^{\text{purple}}_{+}$ characterize position $k+1$ (which will be useful for checking transitions of the Büchi automaton), while $\sigma^{\text{blue}}_{\text{Büchi}}$ and $\sigma^{\text{blue}}_{\text{AP}}$ are Player-Blue strategies that either go to the Büchi part or to the proposition part of the main part of the game.

- Then, using those strategies, we write another SL[BG] formula to enforce that the transitions of the Büchi automaton are correctly applied, following the valuations of the atomic propositions selected in the square states, and that an accepting state is visited infinitely many times.

The construction of the game structure $\mathcal{G}_\Phi$ depicted on Fig. 3 is readily extended to any number of atomic propositions, and to any Büchi automaton. We now explain how we build our SL[BG] formula replacing Formula (1), and ensuring correctness of our reduction.

We do not detail the first step mentioned above and assume that a formula Ψ_{aux} has been written, which properly generates auxiliary strategies, as depicted on Fig. 4 (see [10]). Instead we focus on the Büchi automaton simulation. We look for a strategy of Player Black that will mimic the run of the Büchi automaton, following the valuation of the atomic propositions selected by the square players A_1 to A_n. We also require that the run of the Büchi automaton be accepting.

The formula Ψ enforcing these constraints is as follows[2]:

$$\forall \sigma^{A_1}.\ \exists \sigma^{A_2}.\ \ldots \forall \sigma^{A_{n-1}}.\ \exists \sigma^{A_n}.\ \exists \sigma^{\mathsf{black}}.\ \mathsf{bind}(\sigma^{A_1}, \sigma^{A_2}, \ldots, \sigma^{A_{n-1}}, \sigma^{A_n}, \sigma^{\mathsf{black}}, \sigma^{\mathsf{orange}}_{\infty}).\ \Psi_{\mathsf{aux}}$$

$$\wedge \bigwedge_{p_i,p_j \in \mathsf{AP}} \bigwedge_{q \in Q} (\mathsf{bind}(\sigma^{\mathsf{blue}}_{\text{Büchi}}, \sigma^{\mathsf{purple}}_i)\mathbf{F}\, q) \Leftrightarrow (\mathsf{bind}(\sigma^{\mathsf{blue}}_{\text{Büchi}}, \sigma^{\mathsf{purple}}_j)\mathbf{F}\, q) \qquad (\varphi_1)$$

$$\wedge \bigwedge_{p_i \in \mathsf{AP}} \Big(\big(\mathsf{bind}(\sigma^{\mathsf{blue}}_{\mathsf{AP}}, \sigma^{\mathsf{purple}}).\ \mathbf{F}\, p_i\big) \Rightarrow \big(\mathsf{bind}(\sigma^{\mathsf{blue}}_{\text{Büchi}}, \sigma^{\mathsf{purple}}).\ \bigvee_{q \in Q | p_i \in \mathsf{label}(q)} \mathbf{F}\, q\big) \Big) \qquad (\varphi_2)$$

$$\wedge \bigwedge_{p_i \in \mathsf{AP}} \Big(\big(\mathsf{bind}(\sigma^{\mathsf{blue}}_{\mathsf{AP}}, \sigma^{\mathsf{purple}}).\ \mathbf{F}\ \neg p_i\big) \Rightarrow \big(\mathsf{bind}(\sigma^{\mathsf{blue}}_{\text{Büchi}}, \sigma^{\mathsf{purple}}).\ \bigvee_{q \in Q | p_i \notin \mathsf{label}(q)} \mathbf{F}\, q\big) \Big) \qquad (\varphi_3)$$

$$\wedge \bigwedge_{q \in Q} \mathsf{bind}(\sigma^{\mathsf{blue}}_{\text{Büchi}}, \sigma^{\mathsf{purple}}).\ \mathbf{F}\, q \Rightarrow \bigvee_{q' \in \mathsf{succ}(q)} \mathsf{bind}(\sigma^{\mathsf{blue}}_{+}, \sigma^{\mathsf{purple}}_{+}).\ \mathbf{F}\, q' \qquad (\varphi_4)$$

$$\wedge \mathsf{bind}(\sigma^{\mathsf{blue}}_{\mathsf{acc}}, \sigma^{\mathsf{purple}}_{\mathsf{acc}}).\ \bigvee_{q \in \mathsf{accept}(Q)} \mathbf{F}\, q \qquad (\varphi_5)$$

We now analyze formula Ψ:

- Formula (φ_1) requires that strategy σ^{black} returns the same move after any history of the form $(s, \alpha, \gamma)^k(b, \beta_i, \gamma)$, whichever β_i has been selected by σ^{purple};
- Formulas (φ_2) and (φ_3) constrain the state of the Büchi automaton to correspond to the valuation of the atomic propositions selected. Because of the universal quantification over σ^{purple}, this property will be enforced at all positions and for all atomic propositions;
- Formula (φ_4) additionally requires that two consecutive states of the run of the Büchi automaton indeed correspond to a transition;
- finally, Formula (φ_5) states that for any position (selected by σ^{purple}), there exists a later position (given by $\sigma^{\mathsf{purple}}_{\mathsf{acc}}$) at which the run of the Büchi automaton visits an accepting state.

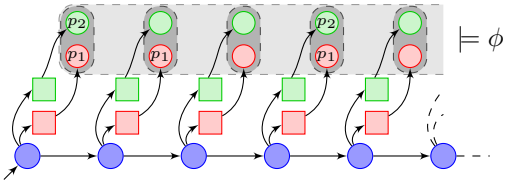
The correctness of the construction is then stated in the next lemma, whose proof can be found in [10].

[2] We notice that Ψ is not syntactically in SL[BG], as some bindings appear before quantifications in Ψ_{aux}. However, quantifiers in Ψ_{aux} could be moved before the bindings of Ψ.

▶ **Lemma 4.** *Formula Φ in* QLTL *is satisfiable if, and only if, Formula Ψ in SL[BG] holds true in state (s, α, γ) of the game $\mathcal{G}_\Phi$.* ◀

▶ Remark. SL[BG] and several other fragments were defined in [32, 33] with the aim of getting more tractable fragments of SL. In particular, the authors advocate for the restriction to *behavioural strategies*: this forbids strategies that prescribe actions depending of what other strategies would prescribe later on, or after different histories. Non-behavioural strategies are thus claimed to have limited interest in practice; moreover, they are suspected of being responsible for the non-elementary complexity of SL model-checking. Our hardness result strengthens the latter claim, as SL[BG] is known for not having behavioral strategies.

▶ Remark. We had to rely on a Büchi automaton instead of directly using the original LTL formula directly in the SL[BG] formula. This is because we need to evaluate the formula not on a real path of our game structure, but on a sequence of "unions" of states. The figure on the right represents this situation for the game structure of Fig. 1: the path on which the LTL formula is given by the red and green circle states, which define the valuations for p_1 and p_2.



4 Periodicity of 1cSL[BG] model checking

In this section we prove our periodicity property for 1cSL[BG]. We inductively define the function $\mathsf{tower} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ as $\mathsf{tower}(a, 0) = a$ and $\mathsf{tower}(a, b+1) = 2^{\mathsf{tower}(a,b)}$. This encodes *towers of exponentials* of the form $2^{2^{\cdot^{\cdot^{\cdot^{a}}}}}$.

▶ **Theorem 5.** *Let $\mathcal{G}$ be a 1cCGS, and φ be a 1cSL[BG] formula. Then there exist a threshold $h \geq 0$ and a period $\Lambda \geq 0$ for the truth value of φ over $\mathcal{G}$. That is, for every configuration (q, c) of $\mathcal{G}$ with $c \geq h$, for every $k \in \mathbb{N}$, $\mathcal{G}, (q, c) \models \varphi$ if, and only if, $\mathcal{G}, (q, c + k \cdot \Lambda) \models \varphi$.*

Furthermore the order of magnitude for $h + \Lambda$ is bounded by

$$\mathsf{tower}\left(\max_{\theta \in \mathsf{Subf}(\varphi)} n_\theta, \max_{\theta \in \mathsf{Subf}(\varphi)} k_\theta + 1\right)^{|Q| \cdot 2^{2^{|\varphi|}}}$$

where $\mathsf{Subf}(\varphi)$ is the set of 1cSL[BG] formulas of φ, k_θ is the number of quantifier alternations in θ, and n_θ is the number of different bindings used in θ.

The rest of this section is devoted to developing the proof of this result, though not with full details. Detailed proofs of intermediate results are given in [10].

We first prove this property for the flat fragment $\mathsf{1cSL}^0[\mathsf{BG}]$, and then extend it to the full 1cSL[BG].

4.1 The flat fragment $\mathsf{1cSL}^0[\mathsf{BG}]$

We fix a 1cCGS $\mathcal{G}$ and a formula $\varphi = Q_1 x_1 \dots Q_k x_k.\ f((\beta_i \phi_i)_{1 \leq i \leq n})$ in $\mathsf{1cSL}^0[\mathsf{BG}]$, where for every $1 \leq j \leq k$, we have $Q_j \in \{\exists, \forall\}$ (assuming quantifiers strictly alternate), f is a Boolean formula over n atoms, and for every $1 \leq i \leq n$, β_i is a complete binding for the players' strategies, and ϕ_i is a 1cLTL formula. We write M for the maximal constant appearing in one of the finite sets describing a counter constraint S appearing in φ.

For every $1 \leq i \leq n$, we let $\mathcal{D}_i$ be a deterministic (counter) parity automaton that recognizes formula ϕ_i (this is the standard LTL-to-(deterministic parity) automata construction in which quantitative constraints are seen as atoms). A run of $\mathcal{G}$ is read in a standard way, with the additional condition that quantitative constraints labelling a state should be satisfied by the counter value when the state is traversed (a state can be labelled by a constraint $\mathsf{cnt} \in S$, with S arbitrarily complex—it does not impact the description of the automaton).

The proof proceeds by showing that, above some threshold, the truth value of φ is periodic w.r.t. counter values. To prove this, we define an equivalence relation over counter values that generates identical strategic possibilities (in a sense that will be made clear later on).

4.1.1 Definition of an equivalence relation

Fix a configuration $\gamma = (\ell, c)$ in $\mathcal{G}$, pick for every $1 \leq i \leq n$ a state d_i in the automaton $\mathcal{D}_i$, and define the tuple $D = (d_1, \ldots, d_n)$. For every context χ_k for variables $\{x_1, \ldots, x_k\}$, we define the level-0 identifier $\mathsf{Id}_{\chi_k}(\gamma, D)$ as:

$$\mathsf{Id}_{\chi_k}(\gamma, D) = \big\{ i \;\big|\; 1 \leq i \leq n \text{ and } \mathsf{out}(\gamma, \beta_i[\chi_k]) \text{ is accepted by } \mathcal{D}_i \text{ from } d_i \big\}$$

where $\beta_i[\chi_k]$ assigns a strategy from χ_k to each player in Agt following β_i.

Assuming we have defined level-$(k-j+1)$ identifiers $\mathsf{Id}_{\chi_{j+1}}(\gamma, D)$ for every partial context χ_{j+1} for variables $\{x_1, \ldots, x_{j+1}\}$, we define the level-$(k-j)$ identifier $\mathsf{Id}_{\chi_j}(\gamma, D)$ for every partial context χ_j for variables $\{x_1, \ldots, x_j\}$ as follows:

$$\mathsf{Id}_{\chi_j}(\gamma, D) = \big\{ \mathsf{Id}_{\chi_{j+1}}(\gamma, D) \;\big|\; \chi_{j+1} \text{ is a context for } \{x_1, \ldots, x_{j+1}\} \text{ that extends } \chi_j \big\}.$$

There is a unique level-k identifier for every configuration $\gamma = (\ell, c)$ and every D, which corresponds to the empty context. It somehow contains full information about what kinds of strategies can be used in the game (this is a hierarchical information set, which contains all level-j identifiers for $j < k$).

Let P be the least common multiple of all the periods appearing in periodic quantitative assertions used in formula φ. We define the following equivalence on counter values:

$$c \sim c' \quad \text{if, and only if,} \quad c = c' \bmod P \text{ and } \forall D.\ \forall \ell.\ \mathsf{Id}_{\emptyset}((\ell, c), D) = \mathsf{Id}_{\emptyset}((\ell, c'), D).$$

Combinatorics. Given a configuration (ℓ, c) and a tuple D, the number of possible values for the level-0 identifier is $\mathsf{tower}(n, 1)$, and for the level-j identifier it is $\mathsf{tower}(n, j+1)$. Hence, the number $\mathsf{ind}_{\sim}$ of equivalence classes of the relation $\sim$ satisfies

$$\mathsf{ind}_{\sim} \leq P \cdot (\mathsf{tower}(n, k+1))^{\left(|Q| \cdot \prod_{1 \leq i \leq n} 2^{2^{|\phi_i|}}\right)} \leq P \cdot (\mathsf{tower}(n, k+1))^{\left(|Q| \cdot 2^{2^{|\varphi|}}\right)}$$

with $|Q|$ the number of states in $\mathcal{G}$. We let $\overline{M} = M + \mathsf{ind}_{\sim} + 1$. By the pigeon-hole principle, there must exist $M < h < h' \leq \overline{M}$ such that $h \sim h'$.

4.1.2 Periodicity property

We define $\Lambda = h' - h$, and now prove that it is a period for φ for counter values larger than or equal to h. Assume that $\gamma = (\ell, c)$ is a configuration such that $c \geq h$, and define $\gamma' = (\ell, c + \Lambda)$ (note that $c + \Lambda \geq h'$). We show that $\mathcal{G}, \gamma \models \varphi$ if, and only if, $\mathcal{G}, \gamma' \models \varphi$.

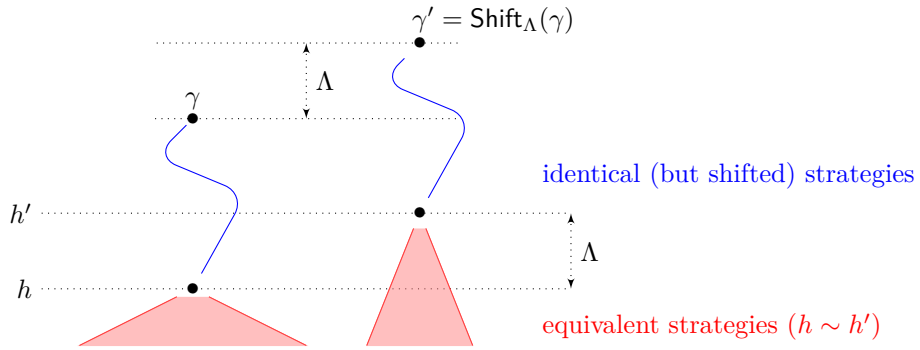


Figure 5 Construction in Lemma 6 (case (ii)).

▶ Notations. For the rest of this proof, we fix the following notations:

1. if ρ is a run starting with counter value $a > c$, then either the counter always remains above c along ρ (in which case we say that ρ is fully above c), or it eventually hits value c, and we define $\rho_{\searrow c}$ for the smallest prefix of ρ such that $\mathsf{last}(\rho_{\searrow c})$ has counter value c;
2. let ρ be a run that is fully above M, and let c be the least counter value appearing in ρ. For every $\nu \geq M - c$, we write $\mathsf{Shift}_\nu(\rho)$ for the run ρ' obtained from ρ by shifting the counter value by ν. It is a real run since the counter values along ρ' are also all above M.
3. if D is a tuple of states of the deterministic automata $\mathcal{D}_i$, and if ρ is a finite run of $\mathcal{G}$ that is fully above M, then we write $D_{+\rho}$ for the image of D after reading ρ.

Let $0 \leq j \leq k$. We assume that χ_j and χ'_j are two contexts for $\{x_1, \ldots, x_j\}$, and D is a tuple of states of the $\mathcal{D}_i$'s. We write $\mathbb{R}^{D,j}_{(\gamma,\gamma')}(\chi_j, \chi'_j)$ if the following property holds for any run ρ from γ:

(i) if ρ is fully above h (or equivalently, if $\rho' = \mathsf{Shift}_{+\Lambda}(\rho)$, which starts from γ', is fully above h'), then for every $1 \leq g \leq j$, $\chi_j(x_g)(\rho) = \chi'_j(x_g)(\rho')$;

(ii) if ρ is not fully above h (equivalently, if $\rho' = \mathsf{Shift}_{+\Lambda}(\rho)$ is not fully above h'), then we decompose ρ (resp. ρ') w.r.t. h (resp. h') and write $\rho = \rho_{\searrow h} \cdot \overline{\rho}$ and $\rho' = \rho'_{\searrow h'} \cdot \overline{\rho}'$. Then:

$$\mathsf{Id}_{\chi_j \overrightarrow{\rho_{\searrow h}}}(\mathsf{last}(\rho_{\searrow h}), \widetilde{D}) = \mathsf{Id}_{\chi'_j \overrightarrow{\rho'_{\searrow h'}}}(\mathsf{last}(\rho'_{\searrow h'}), \widetilde{D})$$

with $\widetilde{D} = D_{+\rho_{\searrow h}} = D_{+\rho'_{\searrow h'}}$. Recall that $\chi_j \overrightarrow{\rho_{\searrow h}}$ shifts all strategies in context χ_j after the prefix $\rho_{\searrow h}$ (that is, χ_j is the strategy such that $\chi_j \overrightarrow{\rho_{\searrow h}}(\pi) = \chi_j(\rho_{\searrow h} \cdot \pi)$ for every π).

We then have:

▶ **Lemma 6.** *Fix $0 \leq j < k$, and assume that $\mathbb{R}^{D,j}_{(\gamma,\gamma')}(\chi_j, \chi'_j)$ holds true. Then:*

1. *for every strategy v for x_{j+1} from γ, one can build a strategy $\mathcal{T}(v)$ for x_{j+1} from γ' such that $\mathbb{R}^{D,j+1}_{(\gamma,\gamma')}(\chi_j \cup \{v\}, \chi'_j \cup \{\mathcal{T}(v)\})$ holds true;*
2. *for every strategy v' for x_{j+1} from γ', one can build a strategy $\mathcal{T}^{-1}(v')$ for x_{j+1} from γ such that $\mathbb{R}^{D,j+1}_{(\gamma,\gamma')}(\chi_j \cup \{\mathcal{T}^{-1}(v')\}, \chi'_j \cup \{v'\})$ holds true.*

Sketch of proof. The idea is the following: either we are in case (1), in which case identical (but shifted) strategies can be applied; or we are in case (2), in which case identical (but shifted) strategies can be applied until counter value h (resp. h') is hit, in which case equality of identifiers allows to apply equivalent strategies. The construction is illustrated in Fig. 5. ◀

We use this lemma to transfer a proof that $\gamma \models_\emptyset \varphi$ to a proof that $\gamma' \models_\emptyset \varphi$. We decompose the proof of this equivalence into two lemmas:

▶ **Lemma 7.** *Fix D^0 for the tuple of initial states of the $\mathcal{D}_i$'s. Assume that $\mathbb{R}^{D^0,k}_{(\gamma,\gamma')}(\chi,\chi')$ holds (for full contexts χ and χ'). Let $1 \leq i \leq n$, and write $\rho = \mathsf{Out}(\gamma, \beta_i[\chi])$ and $\rho' = \mathsf{Out}(\gamma', \beta_i[\chi'])$. Then $\rho \models \phi_i$ if and only if $\rho' \models \phi_i$. In particular, $\gamma \models_\chi f((\beta_i\phi_i)_{1\leq i\leq n})$ if and only if $\gamma' \models_{\chi'} f((\beta_i\phi_i)_{1\leq i\leq n})$.*

Sketch of proof. As long as runs are above h (resp. h') they visit states that satisfy exactly the same atomic properties (atomic propositions and counter constraints), hence they progress in each $\mathcal{D}_i$ along the same run. When value h (resp. h') is hit, they are generated by strategies that have the same level-0 id, which precisely means they are equivalently accepted by each $\mathcal{D}_i$. Hence both outcomes satisfy the same formulas ϕ_i under binding $\beta_i[\chi]$ (resp. $\beta_i[\chi']$). ◀

We finally show the following lemma, by induction on the context, and by noticing that $h \sim h'$ precisely implies the induction property at level 0.

▶ **Lemma 8.** *$\gamma \models_\emptyset \varphi$ if and only if $\gamma' \models_\emptyset \varphi$.*

This allows to conclude with the following corollary:

▶ **Corollary 9.** *Λ is a period for the satisfiability of φ for configurations with counter values larger than or equal to h.*

Furthermore, $h + \Lambda$ is bounded by $M + P \cdot (\mathsf{tower}(n, k+1))^{|Q|\cdot\prod_{1\leq i\leq} 2^{2^{|\varphi|}}} + 1$.

▶ Remark. Note that the above proof of existence of a period, though effective (a period can be computed by computing the truth of identifier predicates), does not allow for an algorithm to decide the model-checking problem. One possible idea to lift that periodicity result to an effective algorithm would be to bound the counter values; however things are not so easy: in Fig. 5, equivalent strategies from h and h' might generate runs with (later on) counter values larger than h or h'. The decidability status of $\mathsf{1cSL}^1[\mathsf{BG}]$ (and of $\mathsf{1cSL[BG]}$) model checking remains open.

4.2 Extension to 1cSL[BG]

We explain how we can extend the previous periodicity analysis to the full logic $\mathsf{1cSL[BG]}$. We fix a formula of $\mathsf{1cSL}^{k+1}[\mathsf{BG}]$

$$\varphi = Q_1 x_1 \dots Q_k x_k \cdot f((\beta_i\phi_i)_{1\leq i\leq n})$$

with the same notations than the ones at the beginning of the previous subsection, but ϕ_i can use closed formulas of $\mathsf{1cSL}^k[\mathsf{BG}]$ as subformulas.

Let Ψ_φ be the set of closed subformulas of $\mathsf{1cSL}^k[\mathsf{BG}]$ that appear directly under the scope of some ϕ_i. We will replace subformulas of Ψ_φ by other formulas involving only (new) atomic propositions and counter constraints. Pick $\psi \in \Psi_\varphi$. Let h_ψ and Λ_ψ be the threshold and the period mentioned in Corollary 9. For every location ℓ of the game, the set of counter values c such that $(\ell, c) \models \psi$ can be written as S^ψ_ℓ (we use a non-periodic set for the values smaller than h_ψ and a periodic set of period Λ_ψ for the values above h_ψ)—note that we know such a set exists, even though there is (for now) no effective procedure to express it. The size of formula S^ψ_ℓ is 1 (we do not take into account the complexity of writing the precise sets used in the constraint). Expand the set of atomic propositions AP with an extra atomic proposition for each location, say p_ℓ for location ℓ, which holds only at location ℓ. For every $\psi \in \Psi_\varphi$, replace that occurrence of ψ in φ by formula $\bigwedge_{\ell\in L} p_\ell \to (\mathsf{cnt} \in S^\psi_\ell)$. This defines formula φ', which is now a $\mathsf{1cSL}^0[\mathsf{BG}]$ formula, and holds equivalently (w.r.t. φ) from every

configuration of $\mathcal{G}$. The size of φ' is that of φ. We apply the result of the previous subsection and get a proof of periodicity of the satisfaction relation for φ', hence for φ.

It remains to compute bounds on the overall period Λ_φ and threshold h_φ. The modulo constraints in φ' involve periods Λ_ψ ($\psi \in \Psi_\varphi$), and the constants used are bounded by h_ψ. So the bound $M_{\varphi'}$ is bounded by $\max(\max_{\psi\in\Psi}(h_\psi), M_\varphi)$ where M_φ is the maximal constant used in φ, and the value $P_{\varphi'}$ is the l.c.m. of the periods used in φ (call it P_φ) and of the Λ_ψ's (for $\psi \in \Psi_\varphi$): hence $P_{\varphi'} \leq P_\varphi \cdot \max_{\psi\in\Psi_\varphi}(\Lambda_\psi)^{|\varphi|}$ Hence for formula φ', we get

$$h_{\varphi'} + \Lambda_{\varphi'} \quad \leq \quad M_{\varphi'} + P_{\varphi'} \cdot \mathsf{tower}\,(n_\varphi, k_\varphi + 1)^{|Q|\cdot 2^{2^{|\varphi'|}}} + 1$$

We infer the following order of magnitude for $h_\varphi + \Lambda_\varphi$, where $\omega_{\Psi_\varphi} = \max_{\psi\in\Psi_\varphi} \omega_\psi$:

$$\begin{aligned}\omega_\varphi &\approx \omega_{\Psi_\varphi} + M_\varphi^{|\varphi|} \cdot (\max \Lambda_\psi)^{|\varphi|} \cdot \mathsf{tower}\,(n_\varphi, k_\varphi + 1)^{|Q|\cdot 2^{2^{|\varphi|}}} \\ &\approx M_\varphi^{|\varphi|} \cdot \omega_{\Psi_\varphi}^{|\varphi|} \cdot \mathsf{tower}\,(n_\varphi, k_\varphi + 1)^{|Q|\cdot 2^{2^{|\varphi|}}}\end{aligned}$$

Using notations of Theorem 5, the order of magnitude can therefore be bounded by

$$\mathsf{tower}\left(\max_{\theta\in\mathsf{Subf}(\varphi)} n_\theta, \max_{\theta\in\mathsf{Subf}(\varphi)} k_\theta + 1\right)^{|Q|\cdot 2^{2^{|\varphi|}}}.$$

▶ Remark. Note that this proof is non-constructive, even for the period and the threshold, since it relies on the model-checking of subformulas, which we don't know how to do. We can nevertheless effectively compute a threshold and a period by taking the l.c.m. of all the integers up to the bound over the period and threshold given in this proof.

5 Conclusion

In this paper, we investigated a quantitative extension of Strategy Logic (and more precisely, of its *Boolean-Goal* fragment) over games played on one-counter games. We proved that the corresponding model-checking problem enjoys a nice periodicity property, which we see as a first step towards proving decidability of the problem. We proved however that, if decidable, the problem is hard; this is proved by showing that model checking the fragment SL[BG] over finite-state games is Tower-hard, hence answering an open question from [32].

We are now trying to see how our periodicity property can be used to prove decidability of the model-checking problem. While such a periodicity property helps getting effective algorithms for model checking CTL over one-counter machines [28], the game setting used here makes things much harder. Other further works also include the more general logic 1cSL, whose decidability status (and complexity) is also open. Finally, we did not manage to extend our hardness proof to turn-based games. It would be nice to understand whether the restriction to turn-based games would make 1cSL[BG] (and SL[BG]) model checking easier.

References

1 R. Alur and D. L. Dill. Automata for modeling real-time systems. In ICALP'90, LNCS 443, pp. 322–335. Springer, 1990.

2 R. Alur and T. A. Henzinger. A really temporal logic. *J. of the ACM*, 41(1):181–203, 1994.

3 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. of the ACM*, 49(5):672–713, 2002.

4 K. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

5 E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In SSC'98, pp. 469–474. Elsevier Science, 1998.

6 A. Bohy, V. Bruyère, E. Filiot, and J.-F. Raskin. Synthesis from LTL specifications with mean-payoff objectives. In TACAS'13, LNCS 7795, pp. 169–184. Springer, 2013.

7 U. Boker, K. Chatterjee, T. A. Henzinger, and O. Kupferman. Temporal specifications with accumulative values. *ACM Transactions on Computational Logic*, 15(4):27:1–27:25, 2014.

8 P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In FORMATS'08, LNCS 5215, pp. 33–47. Springer, 2008.

9 P. Bouyer, P. Gardy, and N. Markey. Quantitative verification of weighted Kripke structures. In ATVA'14, LNCS 8837, pp. 64–80. Springer, 2014.

10 P. Bouyer, P. Gardy, and N. Markey. Weighted strategy logic with boolean goals over one-counter games. Research Report LSV-15-08, Laboratoire Spécification et Vérification, ENS Cachan, France, 2015. 26 pages.

11 T. Brázdil, P. Jančar, and A. Kučera. Reachability games on extended vector addition systems with states. In ICALP'10, LNCS 6199, pp. 478–489. Springer, 2010.

12 S. V. A. Campos and E. M. Clarke. Real-time symbolic model checking for discrete time models. In *Real-time symbolic model checking for discrete time models*, AMAST Series in Computing 2, pp. 129–145. World Scientific, 1995.

13 P. Čermák, A. Lomuscio, and A. Murano. Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In AAAI'15, pp. 2038–2044. AAAI Press, 2015.

14 A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In EMSOFT'03, LNCS 2855, pp. 117–133. Springer, 2003.

15 K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In FSTTCS'10, LIPIcs 8, pp. 505–516. Leibniz-Zentrum für Informatik, 2010.

16 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In CONCUR'07, LNCS 4703, pp. 59–73. Springer, 2007.

17 K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. Research Report 1201.5073, arXiv, 2012.

18 E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In LOP'81, LNCS 131, pp. 52–71. Springer, 1982.

19 E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2000.

20 A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts: Expressiveness and model checking. In FSTTCS'10, LIPIcs 8, pp. 120–132. Leibniz-Zentrum für Informatik, 2010.

21 L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In CONCUR'03, LNCS 2761, pp. 142–156. Springer, 2003.

22 S. Demri and R. Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, 19(6):1541–1575, 2009.

23 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

24 E. A. Emerson, A. K.-L. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4:331–352, 1992.

25 J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *EATCS Bulletin*, 52:244–262, 1994.

26 U. Fahrenberg, L. Juhl, K. G. Larsen, and J. Srba. Energy games in multiweighted automata. In ICTAC'11, LNCS 6916, pp. 95–115. Springer, 2011.

27 S. Göller, C. Haase, J. Ouaknine, and J. Worrell. Model checking succinct and parametric one-counter automata. In ICALP'10, LNCS 6199, pp. 575–586. Springer, 2010.

28 S. Göller and M. Lohrey. Branching-time model checking of one-counter processes. In STACS'10, LIPIcs 20, pp. 405–416. Leibniz-Zentrum für Informatik, 2010.

29 R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

30 F. Laroussinie and N. Markey. Augmenting ATL with strategy contexts. *Inf. & Comp.*, 2015. To appear.

31 F. Laroussinie, N. Markey, and G. Oreiby. Model-checking timed ATL for durational concurrent game structures. In FORMATS'06, LNCS 4202, pp. 245–259. Springer, 2006.

32 F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Transactions on Computational Logic*, 15(4):34:1–34:47, 2014.

33 F. Mogavero, A. Murano, and L. Sauro. A behavioral hierarchy of strategy logic. In CLIMA'14, LNAI 8624, pp. 148–165. Springer, 2014.

34 F. Mogavero, A. Murano, and M. Y. Vardi. Reasoning about strategies. In FSTTCS'10, LIPIcs 8, pp. 133–144. Leibniz-Zentrum für Informatik, 2010.

35 A. Pnueli. The temporal logic of programs. In FOCS'77, pp. 46–57. IEEE Comp. Soc. Press, 1977.

36 J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In SOP'82, LNCS 137, pp. 337–351. Springer, 1982.

37 J. Reichert. On the complexity of counter reachability games. In RP'13, pp. 196–208. Abdulla, Parosh Aziz and Potapov, Igor, 2013.

38 S. Schmitz. Complexity hierarchies beyond elementary. Research Report cs.CC/1312.5686, arXiv, 2013.

39 O. Serre. Parity games played on transition graphs of one-counter processes. In FoSSaCS'06, LNCS 3921, pp. 337–351. Springer, 2006.

40 A. P. Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems*. PhD thesis, Harvard University, Cambridge, Massachussets, USA, 1983. ?

41 A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata, with applications to temporal logic. In ICALP'85, LNCS 194, pp. 465–474. Springer, 1985.

42 W. Thomas. Infinite games and verification (extended abstract of a tutoral). In CAV'02, LNCS 2404, pp. 58–64. Springer, 2002.

43 S. Vester. On the complexity of model-checking branching and alternating-time temporal logics in one-counter systems. In ATVA'15, LNCS, Lecture Notes in Computer Science. Springer, 2015.

44 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Computer Science*, 158(1-2):343–359, 1996.

Decidability in the Logic of Subsequences and Supersequences*

Prateek Karandikar[1] and Philippe Schnoebelen[2]

1 **LIAFA, University Paris Diderot, France**
2 **Laboratoire Specification & Verification (LSV), Cachan, France**

Abstract

We consider first-order logics of sequences ordered by the subsequence ordering, aka sequence embedding. We show that the Σ_2 theory is undecidable, answering a question left open by Kuske. Regarding fragments with a bounded number of variables, we show that the FO^2 theory is decidable while the FO^3 theory is undecidable.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases subsequence, subword, logic, first-order logic, decidability, piecewise-testability, Simon's congruence

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.84

1 Introduction

A subsequence of a (finite) sequence $u = (x_1, \ldots, x_\ell)$ is a sequence obtained from u by removing any number of elements. For example, if $u = (a, b, a, b, a)$ then $u' = (b, b, a)$ is a subsequence of u, a fact we denote with $u' \sqsubseteq u$. Other examples that work for any u are $u \sqsubseteq u$ (remove nothing) and $() \sqsubseteq u$.

In this paper we consider decidability and complexity questions for the first-order logic of finite sequences with the subsequence ordering as the only predicate. The notion of subsequence is certainly a fundamental one in logic, and it occurs prominently in several areas of computer science: in pattern matching (of texts, of DNA strings, etc.), in coding theory, in algorithmics, and in many other areas. We also note that sequences and their subsequences are a special case of a more general notion where a family of finite labelled structures (e.g., trees, or graphs, or ..) are compared via a notion of embedding. Closer to our own motivations, the automatic verification of unreliable channel systems and related problems generate many formulae where the subsequence ordering appears prominently [2, 4, 8, 11].

While decision methods for logics of sequences have been considered in several contexts, the corresponding logics usually do not include the subsequence predicate: they rather consider the prefix ordering, and/or membership in a regular language, and/or functions for taking contiguous subsequences or computing the length of sequences, see, e.g., [10, 7, 1].

As far as we know, Kuske's article [12] is the only one that specifically considers the decidability of the first-order logic of the subsequence ordering *per se*. The article also considers more complex orderings since these decidability questions first occurred in automated

* The first author was partially supported by Tata Consultancy Services and the CEFIPRA Raman-Charpak fellowship. This work was done when he was affiliated to Chennai Mathematical Institute, India and LSV, ENS Cachan, France.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 84–97

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

deduction under the name of *ordered constraints solving* and they involve rather specific orderings on terms and strings [5].

Kuske considers the first-order logic of subsequences over a set of atoms A, denoted $\mathsf{FO}(A^*, \sqsubseteq)$, and notes that the undecidability of its Σ_4 theory can be seen by reinterpreting an earlier undecidability result from [6] for the first-order logic of the lexicographic path ordering. He then shows that already the Σ_3 theory is undecidable even when A contains only two elements, and also shows that the Σ_1 theory is decidable so that the status of the Σ_2 theory remains open.

Our contribution. In this paper we show that the Σ_2 theory of the subsequence ordering is undecidable. On the positive side, we show that the FO^2 theory is decidable (but FO^3 is not). We also prove some complexity bounds for the decidable fragments: the Σ_1 theory is NP-complete and the FO^2 theory is PSPACE-hard.

Outline of the paper. The relevant definitions and basic results are given in section 2. Section 3 develops the reduction that proves undecidability for the Σ_2 and FO^3 theories. Section 4 presents a further reduction that proves undecidability for the Σ_2 theory even when constants are not allowed in the formulae. Then section 5 shows decidability for the two-variable fragment FO^2.

Since our constructions heavily rely on concepts and results from formal language theory, we shall from now on speak of "words", and "letters" (from an "alphabet") rather than sequences and atoms. Note however that the logic $\mathsf{FO}(A^*, \sqsubseteq)$ is defined for any kind of set A.

2 Basic notions

Let $A = \{a_1, a_2, \ldots\}$ be a set called *alphabet*, whose elements are called *letters*. In this paper we only consider finite alphabets for ease of exposition but without any real loss of generality. A *word* is a finite sequence of letters like aac and we use u, v, ..., to denote words, and A^* to denote the set of all words over A. Concatenation of word is written multiplicatively, and ϵ denotes the empty word. We also use regular expressions like $(ab + c)^*$ to denote regular languages (i.e., subsets of A^*). The length of a word u is denoted $|u|$ and, for $a \in A$, we let $|u|_a$ denote the number of occurrences of a in u.

We say that a word u is a *subword* (i.e., a subsequence) of v, written $u \sqsubseteq v$, when u is some $a_1 \cdots a_n$ and v can be written under the form $v_0 a_1 v_1 \cdots a_n v_n$ for some $v_0, v_1, \ldots, v_n \in A^*$. We say a word u is a *factor* of a word v if there exist words v_1 and v_2 such that $v = v_1 u v_2$. For $B \subseteq A$, and $w \in A^*$, we define the projection of w onto B, denoted as $\pi_B(w)$, as the subword of w obtained by removing all letters in $A \backslash B$. For example, $\pi_{\{a,b\}}(abcaccbbc) = ababb$.

We assume familiarity with basic notions of first-order logic as exposed in, e.g., [9]: bound and free occurrences of variables, etc.

In particular, for $n \in \mathbb{N}$, the fragment FO^n consists of all formulae that only use at most n distinct variables (these can have multiple occurrences inside the formula).

The fragments Σ_n and Π_n of $\mathsf{FO}(A^*, \sqsubseteq)$ are defined inductively as follows:

- an atomic formula is in Σ_n and Π_n for all $n \in \mathbb{N}$;
- a negated formula $\neg\phi$ is in Σ_n iff ϕ is in Π_n, it is in Π_n iff ϕ is in Σ_n;
- a conjunction $\phi \wedge \phi'$ is in Σ_n (resp., in Π_n) iff both ϕ and ϕ' are;
- For $n > 0$, an existentially quantified $\exists x \phi$ is in Σ_n iff ϕ is, it is in Π_n iff ϕ is in Σ_{n-1};
- For $n > 0$, a universally quantified $\forall x \phi$ is in Π_n iff ϕ is, it is in Σ_n iff ϕ is in Π_{n-1}.

Note that we do not require formulae to be in prenex normal form when defining the Σ_n and Π_n fragments: for example the formula $\forall x\, \exists y(x \sqsubseteq y \wedge \exists x\, \neg(x \sqsubseteq y))$ is simultaneously in Π_2 and FO^2.

In this article we consider three versions of $\mathsf{FO}(A^*, \sqsubseteq)$, the first-order logic of subsequences over A:

The pure logic: the signature consists of only one predicate symbol, "$\sqsubseteq$", denoting the subword relation. One also uses a countable set $X = \{x, x', y, z, \ldots\}$ of variables ranging over words in A^* and the usual logical symbols.

Note that there is no way in the pure logic to refer to specific elements of A in the logic. However, whether a formula ϕ is true, denoted $\models_{A^*} \phi$, may depend on A (in fact, its cardinality). For example, the closed formula

$$\forall x, y(x \sqsubseteq y \vee y \sqsubseteq x)\,,$$

stating that $\sqsubseteq$ is a total ordering, is true if, and only if, A contains at most one letter.

The basic logic: extends the pure logic by adding all words $u \in A^*$ as constant symbols (denoting themselves). For example, assuming A contains a, b and c, one can write the following sentence:

$$\exists x(ab \sqsubseteq x \wedge bc \sqsubseteq x \wedge abc \not\sqsubseteq x)$$

which is true, as witnessed by the valuation $x \mapsto bcab$.

The extended logic: further allows all regular expressions as unary predicates (with the expected semantics). For these predicates we adopt a more natural notation, writing e.g. $x \in expr$ rather than $P_{expr}(x)$. For example, the extended logic allows writing

$$\forall x\big(\big[\exists y(y \in (ab)^* \wedge x \sqsubseteq y)\big] \Leftrightarrow x \in (a+b)^*\big)$$

which states that the regular language $(a+b)^*$ is the downward closure of $(ab)^*$, i.e., the set of all subwords of its words.

When writing formulae we freely use abbreviations like $x \sqsubset y$ for $x \sqsubseteq y \wedge \neg(y \sqsubseteq x)$ and $x \sqsupseteq y$ for $y \sqsubseteq x$. Note that equality can be defined as an abbreviation since $x \sqsubseteq y \wedge y \sqsubseteq x$ is equivalent to $x = y$. Finally, we use negated symbols as in $x \not\sqsubseteq y$ or $x \notin (ab)^*$ with obvious meaning.

When we write $\mathsf{FO}(A^*, \sqsubseteq)$ without any qualification we refer by default to the basic logic. The pure logic is apparently a very restricted logic, where one may hardly express more than generic properties of the subword ordering like saying that $(A^*, \sqsubseteq)$ is a total ordering, or is a lattice. However, Theorem 3.1 below shows that the pure logic is quite expressive.

We conclude this expository section with

▶ **Theorem 2.1.** *The truth problem for the Σ_1 fragment of $\mathsf{FO}(A^*, \sqsubseteq)$ is NP-complete even when restricting to a fixed alphabet.*

Proof sketch. The upper bound follows from the decidability proof in [12] since it is proved there that a satisfiable quantifier-free formula $\phi(x_1, \ldots, x_n)$ can be satisfied with words of size in $O(n)$ assigned to the x_i's. Guessing linear-sized witnesses $u_1, \ldots, u_n$ and checking that $\models_{A^*} \phi(u_1, \ldots, u_n)$ can be done in NP.

For the lower bound, we reduce from boolean satisfiability. Consider a boolean formula $\phi(x_1, \ldots, x_n)$ over n boolean variables. We reduce it to an $\mathsf{FO}(A^*, \sqsubseteq)$ formula in the Σ_1 fragment

$$\psi \equiv \exists z, x_1, \ldots, x_n(\phi')$$

where ϕ' is obtained from ϕ by replacing each occurrence of x_i with $x_i \sqsubseteq z$ (hence replacing $\neg x_i$ with $x_i \not\sqsubseteq z$). Then, for any alphabet A with at least one letter, ϕ is satisfiable if and only if $\models_{A^*} \psi$. ◀

3 Undecidability for Σ_2

We are interested in solving the *truth problem*. This asks, given an alphabet A and a sentence $\phi \in \mathsf{FO}(A^*, \sqsubseteq)$, whether ϕ is true in the structure $(A^*, \sqsubseteq)$, written $\models_{A^*} \phi$. Restricted versions of the truth problems are obtained for example by fixing A (we then speak of the truth problem *over* A) and/or by restricting to a fragment of the logic.

This section is devoted to proving the following main result.

▶ **Theorem 3.1** (Undecidability). *The truth problem for* $\mathsf{FO}(A^*, \sqsubseteq)$ *is undecidable even when restricted to formulae in the* $\Sigma_2 \cap \mathsf{FO}^3$ *fragment of the basic logic.*

This is done by encoding Post's Correspondence Problem in $\mathsf{FO}(A^*, \sqsubseteq)$. The reduction is described in several stages.

3.1 Expressing simple properties

We start with a list of increasingly complex properties and show how to express them in the basic $\mathsf{FO}(A^*, \sqsubseteq)$ logic. We keep track of what fragment is used, with regards to both the number of distinct variables, and the quantifier alternation depth.

Note that when we claim that a property with m free variables can be expressed in FO^n (necessarily $n \geqslant m$), we mean that the formula only uses at most n variables *including the* m *free variables*.

We let $A = \{a_1, ..., a_\ell\}$ denote an arbitrary alphabet, use B to denote subsets of A, and $a, b, ...$ to denote arbitrary letters from A.

P1. " $x \in B^*$ " can be expressed in $\Sigma_0 \cap \mathsf{FO}^1$: using

$$\bigwedge_{a \in A \setminus B} a \not\sqsubseteq x \,.$$

P2. " $\pi_B(y) \sqsubseteq x$ " can be expressed in $\Pi_1 \cap \mathsf{FO}^3$: building on P1, we use

$$\forall z \big((z \sqsubseteq y \wedge z \in B^*) \implies z \sqsubseteq x \big) \,,$$

noting that $\pi_B(y) \sqsubseteq x$ is equivalent to $\pi_B(y) \sqsubseteq \pi_B(x)$.

P3. " $x = \pi_B(y)$ " can be expressed in $\Pi_1 \cap \mathsf{FO}^3$: building on P1, P2, and using

$$\pi_B(y) \sqsubseteq x \wedge x \sqsubseteq y \wedge x \in B^* \,.$$

P4. " $\pi_B(x) = \pi_B(y)$ " can be expressed in $\Pi_1 \cap \mathsf{FO}^3$: building on P2, and using

$$\pi_B(y) \sqsubseteq x \wedge \pi_B(x) \sqsubseteq y \,.$$

P5. " $x \in aA^*$ ", i.e., " x starts with a ", can be expressed in $\Sigma_2 \cap \mathsf{FO}^3$: building on P1, and using

$$\exists z \Big(a \sqsubseteq z \wedge \Big[\bigwedge_{b \in A \setminus \{a\}} ba \not\sqsubseteq z \Big] \wedge z \sqsubseteq x \wedge \pi_{A \setminus \{a\}}(x) \sqsubseteq z \Big) \,.$$

Here the first two conjuncts require that z contains an occurrence of a and cannot start with another letter. The last two conjuncts require that z is a subword of x which has at least all the occurrences in x of all letters other than a.

Clearly, the mirror property "$x \in A^*a$" can be expressed in $\Sigma_2 \cap \mathsf{FO}^3$ too.

P6. " $x \notin A^*aaA^*$ " can be expressed in $\Sigma_2 \cap \mathsf{FO}^3$: building on P3, and using

$$\exists y \Big(y = \pi_{A\setminus\{a\}}(x) \wedge \forall z \big[(aa \sqsubseteq z \wedge y \sqsubseteq z \wedge z \sqsubseteq x) \implies \bigvee_{b \in A\setminus\{a\}} aba \sqsubseteq z \big] \Big) .$$

Note that this is equivalent to " x does not have aa as a factor". Here $z \sqsubseteq x$ implies that any two occurrences of a in z must come from x. Furthermore, if these are not contiguous in x they cannot be contiguous in z in view of $y = \pi_{A\setminus\{a\}}(x) \sqsubseteq z$.

▶ **Remark 3.2.** Note that the "$y = \pi_{A\setminus\{a\}}(x)$" subformula in P6 uses one variable apart from y and x. We use the same variable name z that is used later in the formula, so that the formula is in FO^3. We similarly reuse variable names whenever possible in later formulae.

P7. " $x \notin A^*BBA^*$ " can be expressed in $\Sigma_2 \cap \mathsf{FO}^3$: as in P6 with

$$\exists y \Big(y = \pi_{A\setminus B}(x) \wedge \forall z \bigwedge_{a,a' \in B} \big[(aa' \sqsubseteq z \wedge y \sqsubseteq z \wedge z \sqsubseteq x) \implies \bigvee_{b \in A\setminus B} aba' \sqsubseteq z \big] \Big) .$$

Note that this is equivalent to " x has no factor in BB ".

P8. " $|\pi_B(x)| = 2$ " can be expressed in $\Sigma_0 \cap \mathsf{FO}^1$: using

$$\Big(\bigvee_{a,a' \in B} aa' \sqsubseteq x \Big) \wedge \bigwedge_{a,a',a'' \in B} aa'a'' \not\sqsubseteq x .$$

3.2 Expressing regular properties

Building on the previous formulae, our next step is to show how any regular property can be expressed in the basic logic by using an enlarged alphabet.

▶ **Lemma 3.3.** *For any regular $L \subseteq A^*$ there is an extended alphabet $A' \supseteq A$ and a formula $\phi_L(x)$ in $\Sigma_2 \cap \mathsf{FO}^3$ over A' such that for all $u \in A'^*$, $u \in L$ if and only if $\models_{A'^*} \phi_L(u)$.*

Proof. Let $\mathcal{A} = (Q, A, \delta, I, F)$ be a NFA recognising L so that $u \in L$ iff $\mathcal{A}$ has an accepting run on input u. We define $\phi_L(x)$ so that it states the existence of such a run, i.e., we put $\phi_L(x) \equiv \exists y\, \psi_{\mathcal{A}}(x, y)$ where $\psi_{\mathcal{A}}(x, y)$ expresses that "y is an accepting run of $\mathcal{A}$ over x."

Let $A' \stackrel{\text{def}}{=} A \cup Q$, assuming w.l.o.g. that A and Q are disjoint. A run $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$ of $\mathcal{A}$ can be seen as a word $q_0a_1q_1a_2 \ldots a_nq_n$ in A'^*. We now define $\psi_{\mathcal{A}}(x, y)$ as the conjunction $\psi_1(x, y) \wedge \psi_2(x, y)$, with

$$\psi_1 \equiv \begin{array}{c} (y \text{ has no factor from } AA) \ \wedge\ (y \text{ has no factor from } QQ) \\ \wedge \ \big(\bigvee_{q \in I} y \text{ begins with } q \big) \ \wedge\ \big(\bigvee_{q \in F} y \text{ ends with } q \big) \ \wedge\ (\pi_A(y) = x) , \end{array}$$

$$\psi_2 \equiv \forall z \left(\begin{array}{c} (x \sqsubseteq z \wedge z \sqsubseteq y \wedge z \text{ has exactly two occurrences of letters from } Q) \\ \implies \Big(\bigvee_{q,q' \in Q} \bigvee_{a,a' \in A} qaa'q' \sqsubseteq z \ \vee \bigvee_{(q,a,q') \in \delta} qaq' \sqsubseteq z \Big) \end{array} \right) .$$

Here ψ_1 reuses simple properties from the previous subsection and states that y is a word alternating between Q (states of $\mathcal{A}$) and A (proper letters), starting with an initial state of $\mathcal{A}$

and ending with an accepting state, hence has the required form $q_0a_1 \dots a_nq_n$. Furthermore, $\pi_A(y) = x$ ensures that y has the form of an accepting run over x. Note that it also ensures $x \in A^*$.

With ψ_2, one further ensures that the above y respects the transition table of $\mathcal{A}$, i.e., that $(q_{i-1}, a_i, q_i) \in \delta$ for $i = 1, \dots, n$. Indeed, assume $z \in A'^*$ satisfies $x \sqsubseteq z \sqsubseteq y$ and contains two occurrences from Q. Thus z is $a_1 \dots a_iq_ia_{i+1}a_{i+2} \dots a_jq_ja_{j+1}a_{j+2} \dots a_n$ for some $1 \leqslant i < j \leqslant n$. If now $j > i + 1$ then z contains $q_ia_{i+1}a_{i+2}q_j$ as a subword and the disjunction after the implication is fulfilled. However, if $j = i + 1$, the only way to fulfil the disjunction is to have $(q_{j-1}, a_j, q_j) \in \delta$.

Finally, $\psi_{\mathcal{A}}(x, y)$ exactly states that y is an accepting run for x and $\models_{A'*} \phi_L(u)$ holds iff $u \in L$. One easily checks that ψ_1 is in $\Sigma_2 \cap \mathsf{FO}^3$, ψ_2 is in $\Pi_1 \cap \mathsf{FO}^3$, so that $\psi_{\mathcal{A}}$ and ϕ_L are in $\Sigma_2 \cap \mathsf{FO}^3$. We reuse variables wherever possible to ensure that only three variables are used (see remark 3.2). For example, the implementation of "y has no factor from QQ" from P7 needs two other variables, and here we use x and z for it. ◀

3.3 Encoding Post's Correspondence Problem

It is now easy to reduce Post's Correspondence Problem to the truth problem for the basic $\mathsf{FO}(A^*, \sqsubseteq)$ logic.

Suppose we have a PCP instance $\mathcal{P}$ consisting of pairs $(u_1, v_1), \dots (u_n, v_n)$ over the alphabet Γ. We let $N = \{1, \dots, n\}$, consider the alphabet $A \stackrel{\text{def}}{=} \Gamma \cup N$, and define

$$\phi_{\mathcal{P}} \equiv \exists x, x' \left(\begin{array}{c} x \in (1u_1 + \dots + nu_n)^+ \ \wedge \ x' \in (1v_1 + \dots + nv_n)^+ \\ \wedge\ \pi_N(x) = \pi_N(x') \wedge \pi_\Gamma(x) = \pi_\Gamma(x') \end{array} \right). \tag{1}$$

Clearly, $\phi_{\mathcal{P}}$ is true iff the PCP instance has a solution.

It remains to check that $\phi_{\mathcal{P}}$ is indeed a formula in the Σ_2 fragment: this relies on Lemma 3.3 for expressing membership in two regular languages, and the P4 properties for ensuring that x and x' contain the same indexes from N and the same letters from Γ. Finally, we note that $\phi_{\mathcal{P}}$ is also a FO^3 formula.

4 Undecidability for the pure logic

In this section we give a stronger version of the undecidability for the Σ_2 fragment.

▶ **Theorem 4.1** (Undecidability for the pure logic). *The truth problem for* $\mathsf{FO}(A^*, \sqsubseteq)$ *is undecidable even when restricted to formulae in the* Σ_2 *fragment of the pure logic.*

The proof is by constructing a Σ_2 formula $\psi(x_1, \dots)$ in the pure logic that defines all the letters and constant words we need to reuse the reduction from the previous section.

Kuske solves the problem in the special case of a formula using only $\{\epsilon, a, b, ab, ba, aa, bb, aba, bab\}$ as constants [12]. We provide a more generic construction whereby all words (up to a fixed length) can be defined in a single Σ_2 formula. One inherent difficulty is that it is impossible to properly define constant words in the pure logic. Of course, with the pure logic one can only define properties up to a bijective renaming of the letters, so $\psi(x_1, \dots)$ will only define letters and words up to renaming. But a more serious problem is that we can only define properties *invariant by mirroring* as we now explain.

For a word $u = a_1a_2 \dots a_\ell$, we let $\widetilde{u}$ denote its mirror image $a_\ell \dots a_2a_1$.

▶ **Lemma 4.2** (Invariance by mirrorring). *If* $\psi(x_1, \dots, x_n)$ *is a formula in the pure logic and* $u_1, \dots, u_n$ *are words in* A^**, then* $\models_{A^*} \phi(u_1, \dots, u_n)$ *if, and only if,* $\models_{A^*} \phi(\widetilde{u_1}, \dots, \widetilde{u_n})$.

Proof Sketch. By structural induction on ϕ, noting that the only atomic formulae in the pure logic have the form $x \sqsubseteq y$, and that $u \sqsubseteq v$ iff $\widetilde{u} \sqsubseteq \widetilde{v}$ for any $u, v \in A^*$. ◀

4.1 Defining letters and short constant words

We now define $\psi(x_1, \ldots)$. In our construction ψ has the form $\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_{13}$ and features a large number of free variables. We describe the construction in several stages, explaining what valuation of its free variables can make ψ true. We start with

$$\forall y(z \sqsubseteq y) \tag{ψ_1}$$
$$\wedge \bigwedge_{1 \leqslant i \neq j \leqslant n} x_i \not\sqsubseteq x_j \tag{ψ_2}$$
$$\wedge \bigwedge_{i=1}^{n} \forall y[y \sqsubseteq x_i \implies (x_i \sqsubseteq y \vee y \sqsubseteq z)] \tag{ψ_3}$$

Here ψ_1 implies $z = \epsilon$, then ψ_2 implies $x_i \neq \epsilon$ so that ψ_3 requires that each x_i is a single letter and furthermore $x_1, \ldots, x_n$ must be different letters as required by ψ_2.

We continue with:

$$\wedge \bigwedge_{i=1}^{n} \big(x_i \sqsubseteq x_i^2 \wedge x_i^2 \not\sqsubseteq x_i \wedge \forall y[y \sqsubseteq x_i^2 \implies (y \sqsubseteq x_i \vee x_i^2 \sqsubseteq y)]\big) \tag{ψ_4}$$

Note that n new free variables, $x_1^2, \ldots, x_n^2$ are involved. First ψ_4 requires that any x_i^2 has at least two letters (it must contain x_i strictly). But it also requires that any subword of x_i^2 is ϵ or x_i or x_i^2, thus x_i^2 has length 2 and can only be $x_i x_i$.

In the same style we introduce new free variables $x_1^3, \ldots, x_n^3$ and $x_1^4, \ldots, x_n^4$ and require that x_i^3 equals $x_i x_i x_i$, and that x_i^4 equals $x_i x_i x_i x_i$ with:

$$\wedge \bigwedge_{i=1}^{n} \big(x_i^2 \sqsubseteq x_i^3 \wedge x_i^3 \not\sqsubseteq x_i^2 \wedge \forall y[y \sqsubseteq x_i^3 \implies (y \sqsubseteq x_i^2 \vee x_i^3 \sqsubseteq y)]\big) \tag{ψ_5}$$
$$\wedge \bigwedge_{i=1}^{n} \big(x_i^3 \sqsubseteq x_i^4 \wedge x_i^4 \not\sqsubseteq x_i^3 \wedge \forall y[y \sqsubseteq x_i^4 \implies (y \sqsubseteq x_i^3 \vee x_i^4 \sqsubseteq y)]\big) \tag{ψ_6}$$

We introduce new free variables $\{y_{i,j}\}_{1 \leqslant i \neq j \leqslant n}$ and conjuncts:

$$\wedge \bigwedge_{1 \leqslant i \neq j \leqslant n} \forall y\,(y \sqsubseteq y_{i,j} \implies y \sqsubseteq z \vee x_i \sqsubseteq y \vee x_j \sqsubseteq y) \tag{ψ_8}$$
$$\wedge \bigwedge_{1 \leqslant i \neq j \leqslant n} \big(x_i \sqsubseteq y_{i,j} \wedge x_j \sqsubseteq y_{i,j} \wedge x_i^2 \not\sqsubseteq y_{i,j} \wedge x_j^2 \not\sqsubseteq y_{i,j}\big) \tag{ψ_9}$$
$$\wedge \bigwedge_{1 \leqslant i \neq j \leqslant n} \big(y_{i,j} \not\sqsubseteq y_{j,i}\big) \tag{ψ_{10}}$$

Here ψ_8 requires that any $y_{i,j}$ only contains letters among x_i and x_j, and ψ_9 requires that it contains exactly one occurrence of x_i and one of x_j. So that $y_{i,j}$ is either $x_i x_j$ or $x_j x_i$. With ψ_{10} we require that $y_{j,i}$ is, among $x_i x_j$ and $x_j x_i$, the word not assigned to $y_{i,j}$.

Now, in view of Lemma 4.2, it is impossible to fix e.g. $y_{i,j} = x_i x_j$. However we can force all $y_{i,j}$ to have "the same orientation". Let i, j, k be three different indexes in $\{1, \ldots, n\}$ and consider the following formula

$$\xi_{i,j,k} \equiv \exists t \left[\begin{array}{lll} & \forall y(y \sqsubseteq t \implies y \sqsubseteq z \vee x_i \sqsubseteq y \vee x_j \sqsubseteq y \vee x_k \sqsubseteq y) & (\xi_1) \\ \wedge & x_i^2 \sqsubseteq t \wedge x_i^3 \not\sqsubseteq t \wedge x_j \sqsubseteq t \wedge x_j^2 \not\sqsubseteq t \wedge x_k \sqsubseteq t \wedge x_k^2 \not\sqsubseteq t & (\xi_2) \\ \wedge & y_{i,j} \sqsubseteq t \wedge y_{j,i} \sqsubseteq t \wedge y_{i,k} \sqsubseteq t \wedge y_{k,i} \not\sqsubseteq t \wedge y_{j,k} \sqsubseteq t \wedge y_{k,j} \not\sqsubseteq t & (\xi_3) \end{array} \right]$$

We claim that, in conjunction with the earlier ψ-conjuncts, $\xi_1 \wedge \xi_2 \wedge \xi_3$ requires $t = x_i x_j x_i x_k$ or $t = x_k x_i x_j x_i$: indeed by ξ_1, t only contains letters among $\{x_i, x_j, x_k\}$, then by ξ_2, t contains exactly 2 occurrences of x_i and exactly one occurrence each of x_j and x_k, then by ξ_3, t has $x_i x_j$ and $x_j x_i$ as subwords, so the single occurrence of x_j is between the two occurrences of x_i and, by ξ_3 again, the occurrence of x_k is outside the two x_i occurrences. Finally, satisfying $\xi_{i,j,k}$ requires $y_{i,k}$ and $y_{j,k}$ to have the same orientation.

We continue the construction of ψ with:

$$\wedge \bigwedge_{1 \leqslant i \neq j \leqslant n} \bigwedge_{k \notin \{i,j\}} \xi_{i,j,k} \qquad (\psi_{11})$$

As just explained, this will force all $y_{i,j}$'s to have the same orientation, i.e., any satisfying assignment will have $y_{i,j} = x_i x_j$ for all i, j, or $y_{i,j} = x_j x_i$ for all i, j.

4.2 Defining long constant words

Once we have defined all words of length 2 (up to mirroring) over the alphabet $\{x_1, \ldots, x_n\}$ (up to renaming), it is easier to systematically define all words of length 3, 4, etc. Actually, we only use constant words of length at most 4 for the formula $\phi_{\mathcal{P}}$ from section 3.

The general strategy relies on a technical lemma we now explain. For $n \in \mathbb{N}$ we say that two words u and v are n-equivalent, written $u \sim_n v$, if u and v have the same set of subwords of length up to n. Thus $\sim_n$ is the piecewise-testability congruence introduced by Simon, see [16, 15].

▶ **Lemma 4.3.** *Let $n \geqslant 2$, and let u and v be words of length $n+1$ with $u \neq v$. Then $u \not\sim_n v$.*

Proof. See appendix. ◀

We can thus introduce new variables $y_{i,j,k}$ and $y_{i,j,k,m}$ for all $i, j, k, m \in \{1, \ldots, n\}$ (allowing repetitions of indexes) and require $y_{i,j,k} = x_i x_j x_k$ and $y_{i,j,k,m} = x_i x_j x_k x_m$, up to mirroring but with the same orientation for all the $y_{i_1,\ldots,i_\ell}$'s. Then we complete the construction of ψ with the following conjuncts:

$$\wedge \bigwedge_{1 \leqslant i,j,k \leqslant n} \text{“}\textit{formula defining } y_{i,j,k}\text{”} \qquad (\psi_{12})$$

$$\wedge \bigwedge_{1 \leqslant i,j,k,m \leqslant n} \text{“}\textit{formula defining } y_{i,j,k,m}\text{”}. \qquad (\psi_{13})$$

In order to require that, for example, $y_{1,5,2} = x_1 x_5 x_2$, it is enough to:

- enumerate all words of length upto 2, and for each say whether it is or is not a subword of $y_{1,5,2}$ ($y_{1,5} \sqsubseteq y_{1,5,2} \wedge x_1^2 \not\sqsubseteq y_{1,5,2} \wedge \ldots$),
- and require that $y_{1,5,2}$ has length 3, by saying that every subword of $y_{1,5,2}$ is itself or is one of the words of length upto 2, and that $y_{1,5,2}$ is distinct from all these words.

The correctness of the construction is guaranteed by Lemma 4.3.

Once all 3-letter words have been defined, we can use them to define 4-letter words (and if needed, 5-letter words, and so on) simlarly, with correctness following from Lemma 4.3.

Finally, we let $\phi'_{\mathcal{P}}$ be obtained from the formula $\phi_{\mathcal{P}}$ —see Eq. (1) page 89— by replacing every constant letter $a_i \in A$ by the variable x_i, and every constant word $a_{i_1} \ldots a_{i_\ell} \in A^*$ by the variable $y_{i_1,\ldots,i_\ell}$ (we use z for the constant word ϵ, and x_i^2 for the constant word $x_i x_i$).

Now we define $\psi_{\mathcal{P}}$ with

$$\psi_{\mathcal{P}} \equiv \exists Z \, (\psi_1 \wedge \cdots \wedge \psi_{13} \wedge \phi'_{\mathcal{P}})$$

where $Z = \{z, x_1, \ldots, x_n, x'_1, \ldots, x'_n, x_1^2, x_1^3, x_1^4, \ldots, y_{1,1}, \ldots, y_{i_1,\ldots,i_\ell}, \ldots\}$ collects all the free variables we used in $\psi_1 \wedge \cdots \wedge \psi_{13}$.

Noting that each ψ_i as well as $\phi'_{\mathcal{P}}$ is a Σ_2 formula, we get that the resulting $\psi_{\mathcal{P}}$ is a Σ_2 formula in the pure logic that is true in $(A^*, \sqsubseteq)$ iff the PCP instance $\mathcal{P}$ is positive. This concludes the proof of Theorem 4.1.

4.3 Undecidability for a fixed alphabet

The above Theorem 4.1 applies to the truth problem *for unbounded alphabet*, i.e., where we ask whether $\models_{A^*} \phi$ for given A and ϕ. In this proof, the alphabet A depends on the PCP instance $\mathcal{P}$ since it includes symbols for the states of the regular automata that define the languages $(1u_1 + \cdots + nu_n)^+$ and $(1v_1 + \cdots + nv_n)^+$ in Eq. (1), and further includes symbols in $N = \{1, \ldots, n\}$.

It is possible to further show undecidability of the Σ_2 fragment even *for a fixed alphabet* A as we now explain. For this we consider a variant of Post's Correspondence Problem:

▶ **Definition 4.4.** The *variant PCP problem* asks, given an alphabet Γ, pairs $(u_1, v_1), \ldots, (u_n, v_n)$ over Γ, and an extra word $w \in \Gamma^*$, whether there exists a sequence $i_1, \ldots, i_\ell$ over $\{1, \ldots, n\}$ such that $w\, u_{i_1} \ldots u_{i_\ell} = v_{i_1} \ldots v_{i_\ell}$.

▶ **Lemma 4.5.** *There is a fixed Γ and a fixed sequence of pairs over Γ for which the variant PCP problem (with only w as input) is undecidable.*

Proof Sketch. One adapts the standard undecidability proof for PCP. Instead of reducing from the question whether a given TM halts, one reduces from the question whether a fixed TM accepts a given input. Note that in the case of a universal TM, the problem is undecidable. Fixing the TM will lead to a fixed sequence of pairs $(u_1, v_1), \ldots, (u_n, v_n)$, and the input of the TM will provide the w parameter of the problem. ◀

▶ **Theorem 4.6** (Undecidability for fixed alphabet)**.** *There exists a fixed alphabet A such that the truth problem for the pure logic $\mathsf{FO}(A^*, \sqsubseteq)$ is undecidable even when restricted to formulae in Σ_2.*

Proof Sketch. We adapt the proof of Theorems 3.1 and 4.1 by reducing from the variant PCP problem with fixed Γ and sequence of pairs. The encoding formula can be

$$\equiv \exists x, x' \left(\begin{array}{c} x \in \Gamma'^* \cdot (1u_1 + \cdots + nu_n)^+ \ \wedge \ x' \in \rho(1v_1 + \cdots + nv_n)^+ \\ \wedge\ \pi_{\Gamma'}(x) = \hat{w} \ \wedge \ \pi_N(x) = \pi_N(x') \wedge \pi_{\Gamma \cup \Gamma'}(x) = \pi_{\Gamma \cup \Gamma'}(x') \end{array} \right) \tag{2}$$

to be compared with Eq. (1). Here we use $\Gamma' = \{\hat{a}, \hat{b}, ..\}$, a renamed copy of $\Gamma = \{a, b, ..\}$, to be able to extract the w prefix in x. The word $\hat{w}$ is simply w from the variant PCP instance with all letters from Γ replaced by corresponding letters from Γ'. We then need to extend the language $(1v_1 + \cdots + nv_n)$ for x' so that letters from Γ' can be used in place of the corresponding letters from Γ. This is done by applying a simple transduction $\rho \stackrel{\text{def}}{=} \left(\bigcup_{a \in \Gamma} \left[\begin{smallmatrix} a \\ a \end{smallmatrix}\right] \cup \left[\begin{smallmatrix} \hat{a} \\ a \end{smallmatrix}\right] \right)^*$.

In the end, we only use two fixed regular languages, and thus a fixed alphabet A. Note however that encoding the input w will require using constant words of unbounded lengths. Here we rely on the fact that our reduction from basic to pure logic can define constant words of arbitrary length in the Σ_2 fragment. ◀

5 Decidability for the FO^2 fragment

In this section we show that for finite alphabets, the truth problem for the 2-variable fragment $\mathsf{FO}^2(A^*, \sqsubseteq)$ is decidable. The proof was first sketched by Kuske [13].

5.1 Rational relations

We recall the basics of rational relations. See [3, Chap. 3] or [14, Chap. 4] for more details.

For finite alphabets A and B, the *rational relations* between A^* and B^* are defined as the subsets of $A^* \times B^*$ recognised by asynchronous transducers. The set of rational relations between A^* and B^* is exactly the closure of the finite subsets of $A^* \times B^*$ under union, concatenation, and Kleene star.

For example, it is easy to see that the subword relation, seen as a subset of $A^* \times A^*$ is a rational relation [3, Example III.5.9], and that the strict subword relation is rational too:[1]

$$\sqsubseteq \;=\; \left(\bigcup_{a\in A}\begin{bmatrix}a\\ \epsilon\end{bmatrix} \cup \begin{bmatrix}a\\ a\end{bmatrix}\right)^*, \qquad\qquad \sqsubset \;=\sqsubseteq \cdot \left(\bigcup_{a\in A}\begin{bmatrix}a\\ \epsilon\end{bmatrix}\right)\cdot \sqsubseteq\ .$$

Define now the *incomparability relation* over A^*, denoted $\perp$, by $u \perp v$ iff $u \not\sqsubseteq v \wedge v \not\sqsubseteq u$.

▶ **Lemma 5.1.** *The incomparability relation over A^* is a rational relation.*

Proof. We cannot simply use the fact that $\not\sqsubseteq$ and $\not\sqsupseteq$ are rational relations since rational relations are not closed under intersection. The way out is to express incomparability as a *union* $\perp = T_1 \cup T_2$ of rational relations, using the following equivalence

$$u \perp v \;\text{ iff }\; \overbrace{(u \not\sqsubseteq v \wedge |u| \leqslant |v|)}^{(u,v)\in T_1} \vee \overbrace{(v \not\sqsubseteq u \wedge |v| \leqslant |u|)}^{(u,v)\in T_2}. \tag{3}$$

The equivalence holds since $|u| > |v|$ implies $u \not\sqsubseteq v$.

We show (see Coro. 5.3) that T_1 is rational. A symmetric reasoning shows that T_2 is rational. This concludes since the union of two rational relations is rational. ◀

In the following proof, we write $w(0:-i]$ to denote the prefix of length $|w|-i$ of an arbitrary word w (assuming $0 \leqslant i \leqslant |w|$).

▶ **Lemma 5.2.** *$(u,v) \in T_1$ iff there exists an integer ℓ, a factorisation $u = a_1 a_2 \dots a_\ell a u'$ of u, and a factorisation $v = v_1 a_1 v_2 a_2 \dots v_\ell a_\ell b v'$ of v such that*

- *$a_1, \dots, a_\ell \in A$ and $v_1, \dots, v_\ell \in A^*$ are such that a_i does not occur in v_i for all $i = 1, \dots, \ell$,*
- *$a, b \in A$ are two letters with $a \neq b$, and*
- *$u', v' \in A^*$ are two suffixes with $|u'| = |v'|$.*

Proof. The ($\Longleftarrow$) direction is clear: the listed conditions guarantee $|u| \leqslant |v|$ and $u \not\sqsubseteq v$.

To see the ($\Longrightarrow$) direction, we assume $(u,v) \in T_1$ and write $u = a_1 \dots a_n$, with $n = |u|$, knowing that $n > 0$ since $u \not\sqsubseteq v$. We say that $i \in \{0, \dots, n\}$ is *good* if $u(0:-i] \sqsubseteq v(0:-i]$, and *bad* otherwise. Clearly, n is good and 0 is bad. Let $m > 0$ be the smallest good index: it is easy to check that taking $\ell = n - m$, $a = a_{\ell+1}$ and $u' = a_{\ell+2} \dots a_n$ proves the claim. ◀

▶ **Corollary 5.3.** *T_1 is a rational relation.*

Proof. Lemma 5.2 directly translates as

$$T_1 = \left(\bigcup_{a\in A}\left[\bigcup_{b\neq a}\begin{bmatrix}b\\ \epsilon\end{bmatrix}\right]^* \cdot \begin{bmatrix}a\\ a\end{bmatrix}\right)^* \cdot \left(\bigcup_{a}\bigcup_{b\neq a}\begin{bmatrix}b\\ a\end{bmatrix}\right) \cdot \left(\bigcup_{a,a'}\begin{bmatrix}a'\\ a\end{bmatrix}\right)^*.$$

◀

[1] When writing such regular expressions we use the vector notation $\left[\begin{smallmatrix}y\\ x\end{smallmatrix}\right]$ to denote (x,y). Note that the domain and the range of the relation correspond to the bottom and, resp., the top, lines of the vectors. We use $\cdot$ to mean concatenation.

5.2 Decidability for FO^2

Let $\mathcal{R} \stackrel{\text{def}}{=} \{=, \sqsubset, \sqsupset, \perp\}$ consists of the following four relations on A^*: equality, strict subword relation, its inverse, and incomparability. These four relations form a partition of $A^* \times A^*$, i.e., for all $u, v \in A^*$, exactly one of $u = v$, $u \sqsubset v$, $u \sqsupset v$, and $u \perp v$ holds.

For any $R \in \mathcal{R}$ and language $L \subseteq A^*$, we define the *preimage* of L by R, denoted $R^{-1}(L)$, as being the language $\{x \in A^* : \exists y \in L : (x, y) \in R\}$. We saw in section 5.1 that each relation $R \in \mathcal{R}$ is rational: we deduce that $R^{-1}(L)$ is regular whenever L is. Furthermore, using standard automata-theoretic techniques, a description of the preimage $R^{-1}(L)$ can be computed effectively from a description of L.

In the following we consider FO^2 formulae using only x and y as variables. We allow formulae to have regular predicates of the form $x \in L$ for fixed regular languages L (i.e., we consider the extended logic). Furthermore, we consider a variant of the logic where we use the binary relations $\sqsubset$, $=$ and $\perp$ instead of $\sqsubseteq$. This will be convenient later. The two variants are equivalent, even when restricting to FO^m or Σ_m fragments: in one direction we observe that $x \sqsubseteq y$ can be defined with $x \sqsubset y \vee x = y$, in the other direction one defines $x \sqsubset y$ with $x \sqsubseteq y \wedge y \not\sqsubseteq x$ and $x \perp y$ with $x \not\sqsubseteq y \wedge y \not\sqsubseteq x$. We also use $x \sqsupset y$ as shorthand for $y \sqsubset x$.

▶ **Lemma 5.4.** *Let $\phi(x)$ be an FO^2 formula with at most one free variable. Then there exists a regular language $L_\phi \subseteq A^*$ such that $\phi(x)$ is equivalent to $x \in L_\phi$. Furthermore, a description for L_ϕ can be computed effectively from ϕ.*

Proof. By structural induction on $\phi(x)$. If $\phi(x)$ is an atomic formula of the form $x \in L$, the result is immediate. If $\phi(x)$ is an atomic formula that uses a binary predicate R from $\mathcal{R}$, the fact that it has only one free variable means that $\phi(x)$ is a trivial $x = x$, or $x \sqsubset x$, or $\ldots$, so that L_ϕ is A^* or $\varnothing$.

For compound formulae of the form $\neg\phi'(x)$ or $\phi_1(x) \vee \phi_2(x)$, we use the induction hypothesis and the fact that regular languages are closed under boolean operations.

There remains the case where $\phi(x)$ has the form $\exists y\, \phi'(x, y)$. We first replace any subformulae of ϕ' having the form $\exists x\, \psi(x, y)$ or $\exists y\, \psi(x, y)$ with equivalent formulae of the form $y \in L_\psi$ or $x \in L_\psi$ respectively, for appropriate languages L_ψ, using the induction hypothesis. Thus we may assume that ϕ' is quantifier-free. We now rewrite ϕ' by pushing all negations inside with the following meaning-preserving transformations:

$$\neg\neg\psi \to \psi \qquad \neg(\psi_1 \vee \psi_2) \to \neg\psi_1 \wedge \neg\psi_2 \qquad \neg(\psi_1 \wedge \psi_2) \to \neg\psi_1 \vee \neg\psi_2$$

and then eliminating negations completely with:

$$\neg(z \in L) \to z \in (A^* \backslash L) \qquad \neg(z_1\ R_1\ z_2) \to z_1\ R_2\ z_2 \vee z_1\ R_3\ z_2 \vee z_1\ R_4\ z_2$$

where R_1, R_2, R_3, R_4 are relations such that $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$. Thus, we may now assume that ϕ' is a positive boolean combination of atomic formulae. We write ϕ' in disjunctive normal form, that is, as a disjunction of conjunctions of atomic formulae. Observing that $\exists y(\phi_1 \vee \phi_2)$ is equivalent to $\exists y\, \phi_1 \vee \exists y\, \phi_2$, we assume w.l.o.g. that ϕ' is just a conjunction of atomic formulae. Any atomic formula of the form $x \in L$, for some L, can be moved outside the existential quantification, since $\exists y(x \in L \wedge \psi)$ is equivalent to $x \in L \wedge \exists y\, \psi$. All atomic formulae of the form $y \in L$ can be combined into a single one, since regular languages are closed under intersection.

Finally we may assume that $\phi'(x, y)$ is a conjunction of a single atomic formula of the form $y \in L$ (if no such formula appears, we can write $y \in A^*$), and some combination of atomic formulae among $x \sqsubset y$, $x \sqsupset y$, $x = y$, and $x \perp y$. If at least two of these appear, then

their conjunction is unsatisfiable, and so $\phi(x)$ is equivalent to $x \in \varnothing$. If none of them appear, $\exists y(y \in L)$ is equivalent to $x \in A^*$ (or to $x \in \varnothing$ if L is empty). If exactly one of them appears, say $x\ R\ y$, then $\exists y\ (y \in L \wedge xRy))$ is equivalent to $x \in L_\phi$ for $L_\phi = R^{-1}(L)$, which is regular as observed earlier. ◀

▶ **Theorem 5.5.** *The truth problem for* $\mathsf{FO}^2(A^*, \sqsubseteq)$ *is decidable.*

Proof. Lemma 5.4 provides a recursive procedure for computing the set of words that make $\phi(x)$ true. When ϕ is a closed formula, this set is A^* or $\varnothing$ depending on whether ϕ is true or not. ◀

5.3 Hardness for FO^2

The main question left open in this paper is the complexity of the decidable FO^2 theory. The recursive procedure described in Lemma 5.4 is potentially non-elementary since nested negations lead to nested complementations of regular languages.

Our preliminary attempts suggest that the question is difficult. At the moment we can only demonstrate the following lower bound.

▶ **Theorem 5.6.** *Truth checking for the basic logic, restricting to* FO^2 *sentences which only use letters (that is, words of length* 1*) as constants, is* PSPACE*-hard.*

Proof. We reduce from TQBF, the truth problem for quantified boolean formulae. W.l.o.g. a given instance of TQBF has the form $\phi' = \exists p_1 \forall p_2 \ldots \exists p_{2n-1} \forall p_{2n} \phi$.

Consider the alphabet A with $4n$ letters, T_i and F_i for each $1 \leqslant i \leqslant 2n$. A word $w \in A^*$ is intended to encode a (partial) boolean valuation V_w of the variables $p_1, \ldots, p_{2n}$: if T_i appears in w, $V_w(p_i) = \mathit{true}$, and if F_i appears in w, $V_w(p_i) = \mathit{false}$. We do not consider "inconsistent" words, in which both T_i and F_i appear. Observe that if x and y represent partial valuations and $x \sqsubseteq y$, then V_y extends V_x. Conversely, any valuation extending V_x can be represented by a suitable y' with $x \sqsubseteq y'$.

For each i, let $\varphi_i(w)$ be a formula that says "the domain of V_w is $\{x_1, \ldots, x_i\}$":

$$\bigwedge_{1 \leqslant j \leqslant i} ((T_j \sqsubseteq w \vee F_j \sqsubseteq w) \wedge \neg(T_j \sqsubseteq w \wedge F_j \sqsubseteq w)) \wedge \bigwedge_{i < j \leqslant 2n} (T_j \not\sqsubseteq w \wedge F_j \not\sqsubseteq w)$$

We now translate the given TQBF instance ϕ' into an FO^2 sentence ψ' in our logic:

$$\begin{aligned}\psi' = \exists x(\varphi_1(x) \wedge \forall y((\varphi_2(y) \wedge x \sqsubseteq y) \implies \exists x(\varphi_3(x) \wedge y \sqsubseteq x \wedge \ldots \\ \wedge \exists x(\varphi_{2n-1}(x) \wedge y \sqsubseteq x \wedge \forall y((\varphi_{2n}(x) \wedge x \sqsubseteq y) \implies \psi)) \ldots)))\end{aligned}$$

where ψ is obtained from ϕ by replacing each p_i with $T_i \sqsubseteq y$.

The formula ψ' uses the two variables x and y alternately, to build up suitable valuations with the appropriate alternation of $\exists$ and $\forall$. It is easy to see that ϕ' is true if and only if ψ' is true.

Finally, it was not necessary to assume that ϕ' had a strict alternation of $\exists$ and $\forall$, but it makes the presentation of the proof simpler. ◀

6 Concluding remarks

We considered the first-order logic of the subsequence ordering and investigated decidability and complexity questions. It was known that the Σ_3 theory is undecidable and that the Σ_1 theory is decidable. We settled the status of the Σ_2 fragment by showing that it has an

undecidable theory, even when restricting to formulae using no constants. To remain in the Σ_2 fragment, our reduction encoded language-theoretic problems rather than undecidable number-theoretic logical fragments as is more usual.

We also showed that the FO^2 theory of the subsequence ordering is decidable using automata-theoretic techniques. The FO^2 fragment is quite interesting. We note that it encompasses modal logics where the subsequence ordering correspond to one step (or its reverse) as used in the verification of unreliable channel systems.

Finally, we provided some new complexity results like Theorems 2.1 and 5.6.

We can list a few interesting directions suggested by this work. First, on the fundamental side, the main question left open is the precise complexity of the FO^2 theory.

Regarding applications, it would be interesting to see how the decidability results can be extended to slightly richer logics (perhaps with some extra functions or predicates, or some additional logical constructs) motivated by specific applications in automated reasoning or program verification.

Acknowledgements. We thank Dietrich Kuske who outlined the proof of Theorem 5.5.

References

1 P. A. Abdulla, M. Faouzi Atig, Yu-Fang Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. String constraints for verification. In *Proc. CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2014.

2 P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

3 J. Berstel. *Transductions and Context-Free Languages*. B. G. Teubner, Stuttgart, 1979.

4 P. Bouyer, N. Markey, J. Ouaknine, Ph. Schnoebelen, and J. Worrell. On termination and invariance for faulty channel machines. *Formal Aspects of Computing*, 24(4–6):595–607, 2012.

5 H. Comon. Solving symbolic ordering constraints. *Int. J. Foundations of Computer Science*, 1(4):387–412, 1990.

6 H. Comon and R. Treinen. Ordering constraints on trees. In *Proc. CAAP '94*, volume 787 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 1994.

7 V. Ganesh, M. Minnes, A. Solar-Lezama, and M. C. Rinard. Word equations with length constraints: What's decidable? In *Proc. HVC 2012*, volume 7857 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2013.

8 Ch. Haase, S. Schmitz, and Ph. Schnoebelen. The power of priority channel systems. *Logical Methods in Comp. Science*, 10(4:4), 2014.

9 J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

10 P. Hooimeijer and W. Weimer. StrSolve: solving string constraints lazily. *Autom. Softw. Eng.*, 19(4):531–559, 2012.

11 P. Karandikar and Ph. Schnoebelen. Generalized Post embedding problems. *Theory of Computing Systems*, 56(4):697–716, 2015.

12 D. Kuske. Theories of orders on the set of words. *RAIRO Theoretical Informatics and Applications*, 40(1):53–74, 2006.

13 D. Kuske. Private email exchanges, April 2014.

14 J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

15 J. Sakarovitch and I. Simon. Subwords. In M. Lothaire, editor, *Combinatorics on words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*, chapter 6, pages 105–142. Cambridge Univ. Press, 1983.

16 I. Simon. Piecewise testable events. In *Proc. 2nd GI Conf. on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

A Proof of Lemma 4.3

Assume $|u| = |v| = n + 1$ and $u \neq v$ as in the statement of the Lemma.

We say that a word w *distinguishes* u *and* v if w is a subword of exactly one of u and v. We have to prove that there exists such a distinguisher w with $|w| \leqslant n$.

Writing a word $w \in A^*$ under the form $w = a_1^{n_1} \dots a_k^{n_k}$ where each a_i is a letter so that $a_i \neq a_{i+1}$ for all $i = 1, \dots, k-1$ and $n_i \geqslant 1$ for all $i = 1, .., k$ is called the *block factorisation* of w. Here k is the number of blocks in w. We now consider several cases:

- Assume that u has only one block. Then $u = a^{n+1}$ for some $a \in A$, and some one-letter word distinguishes u and v. The same reasoning applies if v has only one block.
- Assume that u and v have at least two blocks each, and there is some letter $a \in A$ such that $|u|_a \neq |v|_a$. Then a^k distinguishes u and v for some $k \leqslant n$.
- We are left to deal with cases where u and v have have at least two blocks, and have the same Parikh image, that is, $|u|_a = |v|_a$ for every $a \in A$.
  Assume now that u has exactly two blocks. Then $u \in a^+b^+$ for some $a, b \in A$ with $a \neq b$. Since v has the same number of a's and b's but differs from u, we must have $ba \sqsubseteq v$. But $ba \not\sqsubseteq u$, so ba is a distinguisher (here we use the assumption that $n \geqslant 2$).
- Finally assume that u has at least three blocks. Pick a block B of u which is neither the first nor the last, and let a be the unique letter belonging to B. Let $\ell = |u|_a$ and write u as $u = s_0 a s_1 a \dots a s_\ell$. Then

  $$|s_0| + \dots + |s_\ell| = (n+1) - \ell\,.$$

  At least two of the numbers $|s_0|, \dots, |s_\ell|$ are strictly positive, since the two blocks immediately to the left and right of B both exist, and both do not have a. Thus for all i, $|s_i| < (n+1) - \ell$.
  Since $|v|_a = \ell$, we can write $v = t_0 a t_1 a \dots a t_\ell$. We assume $u \sim_n v$ and obtain a contradiction. For each i such that $0 \leqslant i \leqslant \ell$, consider the word $z_i = a^i s_i a^{\ell - i}$. We have $|z_i| \leqslant n$, and $z_i \sqsubseteq u$. Since $u \sim_n v$, we have $z_i \sqsubseteq v$. Since both z_i and v have exactly ℓ occurrences of a, we have $s_i \sqsubseteq t_i$. This holds for all i, so $u \sqsubseteq v$. But $|u| = |v|$, so $u = v$, which is a contradiction.

Fragments of Fixpoint Logic on Data Words*

Thomas Colcombet[1] and Amaldev Manuel[2]

1 **LIAFA, CNRS, Université Paris 7-Paris Diderot, Paris, France**
`thomas.colcombet@liafa.univ-paris-diderot.fr`
2 **MIMUW, University of Warsaw, Poland**
`amal@mimuw.edu.pl`

Abstract

We study fragments of a μ-calculus over data words whose primary modalities are '*go to next position*' ($\mathtt{X}^g$), '*go to previous position*' ($\mathtt{Y}^g$), '*go to next position with the same data value*' ($\mathtt{X}^c$), '*go to previous position with the same data value* ($\mathtt{Y}^c$)'. Our focus is on two fragments that are called the bounded mode alternation fragment (BMA) and the bounded reversal fragment (BR). BMA is the fragment of those formulas that whose unfoldings contain only a bounded number of alternations between global modalities ($\mathtt{X}^g, \mathtt{Y}^g$) and class modalities ($\mathtt{X}^c, \mathtt{Y}^c$). Similarly BR is the fragment of formulas whose unfoldings contain only a bounded number of alternations between left modalities ($\mathtt{Y}^g, \mathtt{Y}^c$) and right modalities ($\mathtt{X}^g, \mathtt{X}^c$). We show that these fragments are decidable (by inclusion in Data Automata), enjoy effective Boolean closure, and contain previously defined logics such as the two variable fragment of first-order logic and DataLTL. More precisely the definable language in each formalism obey the following inclusions that are effective.

$$\mathrm{FO}^2 \subsetneq \text{DataLTL} \subsetneq \mathrm{BMA} \subsetneq \mathrm{BR} \subsetneq \nu \subseteq \text{Data Automata} .$$

Our main contribution is a method to prove inexpressibility results on the fragment BMA by reducing them to inexpressibility results for combinatorial expressions. More precisely we prove the following hierarchy of definable languages,

$$\emptyset = \mathrm{BMA}^0 \subsetneq \mathrm{BMA}^1 \subsetneq \cdots \subsetneq \mathrm{BMA} \subsetneq \mathrm{BR} ,$$

where BMA^k is the set of all formulas whose unfoldings contain at most $k-1$ alternations between global modalities ($\mathtt{X}^g, \mathtt{Y}^g$) and class modalities ($\mathtt{X}^c, \mathtt{Y}^c$). Since the class BMA is a generalisation of FO^2 and DataLTL the inexpressibility results carry over to them as well.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Data words, Data automata, Fixpoint logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.98

1 Introduction

Data words are words of the form $(a_1, d_1) \dots (a_n, d_n) \in (\Sigma \times \mathcal{D})^*$ where Σ is a finite set of letters and $\mathcal{D}$ is an infinite domain of *data values*. Typically the alphabet Σ abstracts a finite set of actions or events and the set of data values $\mathcal{D}$ models some sort of identity information. Thus, data words can model a number of scenarios where the information is linearly ordered and it is composed of finite as well as unbounded elements. For example the authors of [1] imagine Σ as the actions of a finite program and $\mathcal{D}$ as process ids. Then, an execution trace

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454.

of a system with unbounded instances of the program can be modeled as a data word in which each action is associated with the identifier of the process which has generated it.

The paradigmatic question in the study of data words is to develop suitable models (in particular automata and logics) to specify properties of data words. Sure enough there exists a rich variety of models for specifying properties of data words that includes Data Automata [4], Register Automata [14, 9], Pebble Automata [18], Class Memory Automata [1], Class Automata [2], Walking Automata [17], Variable Automata [12], First-Order logic with two variables [4], guarded MSO logic [5], DataLTL [15], Freeze-Logics[9, 13], Logic of Repeating Values [8], XPath [10, 11], Regular expressions [16], Data Monoids [3] etc.

In this work we further study a modal fixpoint logic on data words that we introduced in [6]. This logic is composed of four modalities that allow to evaluate formulas on the successor, class successor (the nearest future position with the same data value), predecessor and class predecessor (nearest past position with the same data value) positions, $\mathtt{Y}^g, \mathtt{Y}^c$. In addition there is a couple of zeroary modalities that describes whether these positions coincide or not. To build the formulas, besides the usual Boolean operations, it is allowed to form the least and greatest fixpoints of formulas . In [6] it is shown that the satisfiability problem for the set of formulas that use only least fixpoints is undecidable, whereas the fragment that consists of only greatest fixpoints is subsumed by Data Automata and hence it has a decidable satisfiability problem. The main result of the work was the decidability of an alternation-free fragment of the logic that further bounds the number of change of directions in evaluating the formulas by using a generalisation of Data Automata.

Contributions

In the present paper, we aim at restricting the power of the above μ-calculus logic for data words for obtaining classes that are closed under all Boolean connectives, mirroring, and enjoy decidability of emptiness and universality. We consider two restricted fragments that achieve this goal. The first one, called BMA (for Bounded Mode Alternation) syntactically bounds the number of changes between class and global modes. The second, called BR (for Bounded Reversal), syntactically bounds the number of changes between left modalities and right modalities.

It is easy to show that BMA is contained in Data Automata. It is not very difficult to show that BMA is subsumed by BR, that is to say for every formula in BMA there is an equivalent one in BR. Our main result is the strictness of this last inclusion, i.e. that there is a formula in BR for which there is no equivalent formula in BMA. This proof uses a deep result from combinatorics called the Hales-Jewett theorem. As a proof device we use a sort of circuits called combinatorial expressions that were introduced in [7]. These expressions define functions over an infinite domain (for instance the integers). They are built by composing *gates* that are functions of two kinds, either the function has a bounded arity, or the function has a bounded domain. In [7] it is shown that certain properties (a property is a function that has a binary codomain) for instance *the given sequence of positive integers has gcd 1* or *the given sequence of integers sum to 0* cannot be computed by expressions of fixed depth. We use a variant of this theorem in this paper to show that there is a formula in BR for which there is no equivalent one in BMA. More precisely it is shown that there is a specific formula in BR such that if it has an equivalent formula in BMA, then it is possible to construct expressions of fixed depth for a particular property and since that particular property cannot be computed by fixed depth expressions, we derive a contradiction. One thing to note is that since the techniques developed in [7] are general enough to derive impossibility results for a large family of properties, correspondingly the proof method developed here can be used to show inexpressibility results for a variety of formulas.

Now we examine the implications of our result in a larger context. As mentioned earlier, the results mentioned here have very close connection with *Data Automata* (DA for short). The well known feature of DA is that it subsumes the logic $\mathrm{FO}^2(\Sigma, <, +1, \sim, +^c 1)$ on data words where Σ denotes the unary predicates indicating the letters, $<$ is the linear order on positions, $+1$ is the successor relation on positions, $\sim$ is the equivalence relation on positions with respect to the data values (i.e. $i \sim j$ if $d_i = d_j$), and $+^c 1$ is the class successor relation. It is known that data languages recognisable by DA are closed under union, intersection and letter-to-letter projection, but not under complementation [4]. Since FO^2 formulas are closed under Boolean operations, it is evident that Data Automata strictly subsumes the logic FO^2. This observation prompts the question that if there are other classes subsumed by DA that are closed under Boolean operations. The fragments introduced in the paper answer this question positively. Note only that, there are automata theoretic characterisations that are natural variants of DA for both these fragments (we only present the one for BMA).

Another and perhaps more important question is how to show that a given data language is not expressible in FO^2. Note that in some cases, using the techniques on words over finite alphabets it is possible to show that a given data language is not definable in FO^2 (for instance to show that data words of even length are not definable in FO^2). We are interested in those cases where such reductions are not possible, in particular where the property given is dependent on the data values. We don't have a complete solution to this problem yet, but our method to prove inexpressibility results on BMA offers a partial answer. This is because the logic $\mathrm{FO}^2(\Sigma, <, +1, \sim, +^c 1)$, as it is shown in this paper, is equivalent to the unary fragment of a temporal logic, namely DataLTL [15], which is a strict subfragment of BMA. DataLTL is the temporal logic where usual temporal operators such as until, future, past etc. exist both on the linear order on positions (called the global order) as well as on the suborders formed by subsets of positions that share the same data value (called the *class orders*). For instance the temporal operator $\mathtt{F}^g\varphi$ is true a position if there is a position in the future that satisfies the formula φ, whereas the formula $\mathtt{F}^c\varphi$ is true at a position if there is a future position that has the same data value as the current position and that satisfies the formula φ. The unary fragment of DataLTL is the subclass of formulas that uses only the unary temporal operators (such as $\mathtt{F}^g, \mathtt{P}^g, \mathtt{F}^c$ etc). Since every such operator is expressible in FO^2 it is immediate that unary DataLTL is subsumed by the logic FO^2. But the converse direction, which is shown in the paper, is not obvious, since it is not immediate how to translate formulas like $\exists y\,(a(x) \wedge b(y) \wedge x < y \wedge x \not\sim y)$. Thus inexpressibility results on the fragment BMA renders directly corresponding results on all sublogics including FO^2 and DataLTL.

Finally let us also note that the translations outlined in this paper, namely

$$\mathrm{FO}^2 \subsetneq \text{DataLTL} \subsetneq \text{BMA} \subsetneq \text{BR} \subsetneq \nu \subseteq \text{DA},$$

constitutes an alternate proof the main result of [4] that FO^2 is subsumed by DA. The proof in [4] is a direct translation of FO^2 formulas by a intricate case analysis. Our proof, however, is modular and makes use of analogous constructions from automata theory on finite words.

Related work

As mentioned already this work is strongly related to DataLTL, FO^2 and DA. One other very popular ecosystem on data words is that of Register Automata and the associated logics such as Freeze LTL, Freeze μ-calculus, Xpath [9, 13, 8, 10, 11] etc. Our inexpressibility result implies that BMA is incomparable to Register Automata (in particular nondeterministic 1-Register Automata). Since all our modalities are expressible in terms of successor, predecessor,

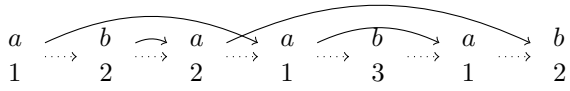


Figure 1 A data word and its graph. Dotted and thick arrows denote the successor and class successor relations respectively.

freeze operator and fixpoint operators, our fixpoint logic is subsumed by Freeze μ-calculus of [13]. However it should be noted that the latter logic is highly undecidable [9]. The decidable fragment of Freeze μ-calculus (and also Freeze LTL) is unidirectional (only future modalities) but our logic is naturally two-way. Finally the decidable two-way fragment of Freeze LTL, namely Simple Freeze LTL is equivalent to FO^2 and hence it is subsumed by BMA. Therefore our method of proving inexpressibility extends to this logic as well.

Structure of the document

In Section 2 we present the definition of our fixpoint logic and give some examples. In Section 3 we recall the *composition* operator (*comp*) on sets of formulas and define the fragments BMA and BR using it. Thereafter, a characterisation of the class BMA in terms of cascades of automata, that is used in the proof of the separation theorem, is given. In Section 4, first we recollect the paradigm of combinatorial expressions and state the necessary results for our purpose. Afterwards it shown how to translate a cascade on data words with a specific structure to expressions and the separation theorem is proved. In Section 5 we conclude.

2 μ-Calculus on Data Words

In this section, we recall the basics of the μ-calculus on data words [6].

Fix an infinite set $\mathcal{D}$ of *data values*. *Data words* are words of the form

$$u = (a_1, d_1) \cdots (a_n, d_n) \in (\Sigma \times \mathcal{D})^*$$

where Σ is a finite *alphabet* of *letters*. Indices in a word are called *positions*. A maximal set of positions in u with the same data value is called a *class*. The set of classes in u define an equivalence relation $\sim$, called the *class relation*, on the set of positions of u. Given a permutation σ of $\mathcal{D}$, it can be applied on a data word as expected, yielding $\sigma(u) = (a_1, \sigma(d_1)) \dots (a_n, \sigma(d_n))$. The data words u and $\sigma(u)$ have the same class relation. A *data language* is a set of data words that is invariant under such applications of the permutations of $\mathcal{D}$.

For our purposes, it is convenient to see data words as graphs in the following manner. To each data word $w = (a_1, d_1) \dots (a_n, d_n) \in (\Sigma \times \mathcal{D})^*$ associate the graph $G_w = ([n], \ell, +1, +^c1)$ where $[n]$ is the set of positions $\{1, \dots, n\}$, $\ell : \Sigma \to 2^{[n]}$ is the labelling function $\ell(a) = \{i \mid a_i = a\}$, the binary relation $+1$ denotes the *successor* relation on positions, *i.e.*, $+1(i, j)$ if $j = i + 1$, and the binary relation $+^c1$ denotes the *class successor* relation on positions, *i.e.*, $+^c1(i, j)$ if $i < j$, $d_i = d_j$, and $d_m \neq d_i$ for all $i < m < j$. We call *predecessor* relation (*resp.*, *class predecessor* relation) the reverse of the successor relation (*resp.*, class successor relation). We implicitly identify a data word with its graph. Figure 1 shows a data word and its corresponding graph.

Seen as such graphs, data words are naturally prone to the use of temporal logics. Let $\mathit{Prop} = \{p, q, \dots\}$ and $\mathit{Var} = \{x, y, \dots\}$ be countable sets of *propositional variables* and

$$
\begin{aligned}
[\![fst^g]\!]_w &= \{1\} & [\![\mathtt{X}^g\varphi]\!]_w &= [\![\varphi]\!]_w - 1\\
[\![lst^g]\!]_w &= \{n\} & [\![\mathtt{Y}^g\varphi]\!]_w &= [\![\varphi]\!]_w + 1\\
[\![fst^c]\!]_w &= \{i \mid \nexists j = i -^c 1\} & [\![\mathtt{X}^c\varphi]\!]_w &= [\![\varphi]\!]_w -^c 1\\
[\![lst^c]\!]_w &= \{i \mid \nexists j = i +^c 1\} & [\![\mathtt{Y}^c\varphi]\!]_w &= [\![\varphi]\!]_w +^c 1\\
[\![\varphi_1 \wedge \varphi_2]\!]_w &= [\![\varphi_1]\!]_w \cap [\![\varphi_2]\!]_w & [\![\mathsf{S}]\!]_w &= \{i \mid i+1 = i +^c 1\}\\
[\![\varphi_1 \vee \varphi_2]\!]_w &= [\![\varphi_1]\!]_w \cup [\![\varphi_2]\!]_w & [\![\mathsf{P}]\!]_w &= \{i \mid i-1 = i -^c 1\}\\
[\![\mu x.\varphi]\!]_w &= \cap\{S \subseteq [n] \mid [\![\varphi]\!]_{w[\ell(x):=S]} \subseteq S\} & [\![p]\!]_w &= \ell(p)\\
[\![\nu x.\varphi]\!]_w &= \cup\{S \subseteq [n] \mid S \subseteq [\![\varphi]\!]_{w[\ell(x):=S]}\} & [\![\neg p]\!]_w &= [n] \setminus \ell(p)\\
[\![x]\!]_w &= \ell(x)
\end{aligned}
$$

Figure 2 Semantics of μ-calculus on data words $w = ([n], +1, +^c 1, \ell)$.

fixpoint variables respectively. The *μ-calculus* on data words is the set of all *formulas* φ respecting the following syntax:

$$\varphi := x \mid A \mid \neg A \mid \mathtt{M}\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

where $\mathtt{M} := \mathtt{X}^g \mid \mathtt{X}^c \mid \mathtt{Y}^g \mid \mathtt{Y}^c$ and $A := p \in Prop \mid \mathsf{S} \mid \mathsf{P} \mid fst^c \mid fst^g \mid lst^c \mid lst^g$

The elements of $\mathtt{M}$ are called *modalities*, and the ones of A, *atoms*. The set of zeroary modalities $\{fst^c, fst^g, lst^c, lst^g, \mathsf{P}, \mathsf{S}\}$ will be denoted by the symbol Z for the rest of the paper.

The semantic of a formula φ, over a data word w is the set of positions of w where "φ is true" (denoted as $[\![\varphi]\!]_w$). The formal definition is given in Figure 2. The different constructs have their expected meaning, keeping in mind that the class modalities $\mathtt{X}^c, \mathtt{Y}^c, fst^c, lst^c$ have to be interpreted on the word restricted to the current data value. The modality S (*resp.*, P) holds at a position i if the successor and class successor i coincide (*resp.* the predecessor and class predecessor coincide).

Note that in this definition of the logic, negations in a formula are located at the leaves. It is nevertheless possible, as usual, to negate such formulas by pushing the negation toward the leaves, but this requires a bit of care when negating modalities and fixpoints. For instance, $\neg\mathtt{X}^c\varphi$ is not equivalent to $\mathtt{X}^c\neg\varphi$, but to $lst^c \vee \mathtt{X}^c\neg\varphi$. Similar arguments have to be used for all modalities. Following these ideas, we define the *dual* modalities $\tilde{\mathtt{X}}^g\varphi \equiv lst^g \vee \mathtt{X}^g\varphi$, $\tilde{\mathtt{Y}}^g\varphi \equiv fst^g \vee \mathtt{Y}^g\varphi$, $\tilde{\mathtt{X}}^c\varphi \equiv lst^c \vee \mathtt{X}^c\varphi$ and $\tilde{\mathtt{Y}}^c\varphi \equiv fst^c \vee \mathtt{Y}^c\varphi$. These modalities are considered dual since $\tilde{\mathtt{X}}^g\varphi \equiv \neg\mathtt{X}^g\neg\varphi$, ... Similarly $\mu x.\varphi(x) \equiv \neg\nu x.\neg\varphi(\neg x)$.

Next we lay out some terminology and abbreviations which we will use in the subsequent sections. Let λ denote either μ or ν. Every occurrence of a fixpoint variable x in a subformula $\lambda x.\psi$ of a formula is called *bound*. All other occurrences of x are called *free*. A formula is called a *sentence* if all the fixpoint variables in φ are bound. If $\varphi(x_1, \ldots, x_n)$ is a formula with free variables $x_1, \ldots, x_n$, then by $\varphi(\psi_1, \ldots, \psi_n)$ we mean the formula obtained by substituting ψ_i for each x_i in φ. As usual the bound variables of $\varphi(x_1, \ldots, x_n)$ may require a renaming to avoid the capture of the free variables of ψ_i's. For a sentence φ and a position i in the word w, we denote by $w, i \models \varphi$ if $i \in [\![\varphi]\!]_w$. The notation $w \models \varphi$ abbreviates the case when $i = 1$. The data language of a sentence φ, denoted as $L(\varphi)$, is the set of data words w such that $w \models \varphi$.

By *μ-fragment* we mean the subset of μ-calculus which uses only μ-fixpoints. Similarly *ν-fragment* stands for the subset which uses only ν-fixpoints.

▶ **Example 1** (temporal modalities). An example of a formula would be $\varphi \, \mathtt{U}^g \, \psi$ which holds if ψ holds in the future, and φ holds in between. This can be implemented as $\mu x.\psi \vee (\varphi \wedge \mathtt{X}^g x)$ The formula $\varphi \mathtt{U}^c \psi = \mu x.\psi \vee (\varphi \wedge \mathtt{X}^c x)$ is similar, but for the fact that it refers only to the class of the current position. The formula $\mathtt{F}^g \varphi$ abbreviates $\top \, \mathtt{U}^g \, \varphi$, and its dual is $\mathtt{G}^g \varphi = \neg \mathtt{F}^g \neg \varphi$. The constructs $\mathtt{S}^g$, $\mathtt{S}^c$, $\mathtt{P}^g$, $\mathtt{P}^c$, $\mathtt{H}^g$ and $\mathtt{H}^c$, are defined analogously, using past modalities, and correspond respectively to $\mathtt{U}^g$, $\mathtt{U}^c$, $\mathtt{F}^g$, $\mathtt{F}^c$, $\mathtt{G}^g$ and $\mathtt{G}^c$. For instance, $\mathtt{F}^c \mathtt{P}^c \varphi$ expresses that there is a position in the class that satisfies φ and $\mathtt{F}^c \mathtt{P}^c (\varphi \wedge \tilde{\mathtt{X}}^c \mathtt{G}^c \neg \varphi \wedge \tilde{\mathtt{Y}}^c \mathtt{H}^c \neg \varphi)$ expresses that there exists exactly one position which satisfies φ in the class.

3 The bounded reversal and bounded mode alternation fragments

In this section we introduce the bounded mode alternation and bounded reversal fragments (BMA and BR) and compare these two fragments between themselves and with other logics (Theorem 5).

3.1 Definition of the fragments

Before delving into the technical details let us outline the intuition behind each of the fragments. The four modalities $\mathtt{X}^g, \mathtt{Y}^g, \mathtt{X}^c$ and $\mathtt{Y}^c$ can be divided along two axis. Based on the directions: there are the *left modalities* $\mathtt{Y}^g$, $\mathtt{Y}^c$, and *right modalities* $\mathtt{X}^g$, $\mathtt{X}^c$. Based on the modes: there are *global modalities* $\mathtt{X}^g$, $\mathtt{Y}^g$, and *class modalities* $\mathtt{X}^c$, $\mathtt{Y}^c$. The BR and BMA fragments are defined by limiting the number of alternation between this types of modalities. This is formally achieved using the operation *comp* that we define now.

Let Ψ be a set of μ-calculus formulas. Define the sets $comp^i(\Psi)$ for $i \in \mathbb{N}$ inductively

- $comp^0(\Psi) = \emptyset$,
- $comp^{i+1}(\Psi) = \{\psi(\varphi_1, \dots, \varphi_n) \mid \psi(x_1, \dots, x_n) \in \Psi,\ \varphi_1, \dots, \varphi_n \in comp^i(\Psi)\}$ in which the substitution $\psi(\varphi_1, \dots, \varphi_n)$ is allowed only if none of the free variables of $\varphi_1, \dots, \varphi_n$ get bound in $\psi(\varphi_1, \dots, \varphi_n)$.

The set of formulas $comp(\Psi)$ is defined as $comp(\Psi) = \bigcup_{i \in \mathbb{N}} comp^i(\Psi)$. For a formula $\psi \in comp(\Psi)$, the *comp-height* of ψ in $comp(\Psi)$ in the least i such that ψ is in $comp^i(\Psi)$.

We are now ready to define the BR and BMA fragments of the μ-calculus. For a set of modalities M, define $formulas(M)$ to be the set of formulas that uses only the modalities M apart from the zeroary modalities.

▶ **Definition 2.** The BMA and the BR fragments of μ-calculus are respectively:

$$\text{BMA} = comp\left(formulas\left(\{\mathtt{X}^g, \mathtt{Y}^g\}\right) \cup formulas\left(\{\mathtt{X}^c, \mathtt{Y}^c\}\right)\right),$$
$$\text{and} \qquad \text{BR} = comp\left(formulas\left(\{\mathtt{X}^g, \mathtt{X}^c\}\right) \cup formulas\left(\{\mathtt{Y}^g, \mathtt{Y}^c\}\right)\right).$$

Further, BMA^k denotes the subset of BMA with comp-height k. Similarly BR^k stands for the subset of BR with comp-height k.

▶ **Example 3.** Let

$$\varphi_1 = \nu x.(\mathtt{X}^c x \vee \mathtt{X}^g \mu y.(q \wedge \mathtt{Y}^c y)), \qquad \varphi_2 = \nu x.\left(\mathtt{X}^c lst^g \vee \mathtt{X}^c \mathtt{Y}^g x\right),$$
$$\varphi_3 = \mu x.((\nu y.\, q \vee \mathtt{X}^c y) \vee \mathtt{X}^g x \vee \mathtt{Y}^g x), \qquad \text{and} \qquad \varphi_4 = \mu x.(\mathtt{X}^c \mathtt{X}^g x \vee p).$$

The formula φ_1 is in BR^2 and in BMA^3. The formula φ_2 is neither in BR nor in BMA. The formula φ_3 is in BMA^2 but not in BR. The formula φ_4 is in BR^1 but not in BMA.

▶ **Example 4.** Consider the language L_k that contains the data words such that, by applying k-times the sequence of the global successor followed by the class successor, one reaches a position labeled with letter a. The language L is the union of all L_k for k ranging over all non-negative integers. The language L_k is defined by φ_k and L by φ defined as follows:

$$\varphi_k = \overbrace{\mathtt{X}^g\mathtt{X}^c \dots \mathtt{X}^g\mathtt{X}^c}^{k\text{-times}} a, \qquad \text{and} \qquad \varphi = \mu x.(\mathtt{X}^g\mathtt{X}^c x \vee a) \ .$$

The formula φ_k is in BR^1 and in BMA^{2k}. The formula φ is in BR^1, but not in BMA. Later in Section 4 we will prove that a variant of L is not definable by any formula in BMA.

Let us now state the main theorem of this section, namely the inclusions between the fragments of the μ-calculus in terms of the data languages defined. Below *DataLTL* denotes the temporal logic on data words consisting of the modalities $\{\mathsf{S}, \mathsf{P}, \mathtt{X}^g, \mathtt{Y}^g, \mathtt{X}^c, \mathtt{Y}^c, \mathtt{U}^g, \mathtt{S}^g, \mathtt{U}^c, \mathtt{S}^c\}$, *uDataLTL* is the unary sublogic consisting of the modalities $\{\mathsf{S}, \mathsf{P}, \mathtt{X}^g, \mathtt{X}^c, \mathtt{Y}^g, \mathtt{Y}^c, \mathtt{F}^c, \mathtt{F}^g, \mathtt{P}^g, \mathtt{P}^c\}$ and ν denotes the fragment of the μ-calculus containing only the greatest fixpoints (ν's).

▶ **Theorem 5.** *The following inclusions hold for definable languages,*

$$\mathrm{FO}^2(\Sigma, <, +1, \sim, +^c 1) = \mathit{uDataLTL} \subsetneq \mathit{DataLTL} \subsetneq \mathrm{BMA} \subseteq \mathrm{BR} \subsetneq \nu \subseteq \mathrm{DA} \ .$$

3.2 Characterising BMA as cascades of automata

Next we give a characterisation of BMA in terms of cascades of finite state automata. It is classical that composition (*comp*) corresponds to the natural operation of composing sequential transducers that compute subset of subformulas that are true at each position. Given a μ-calculus formula φ over words, we can see it as a transducer that reads the input, and labels every position with one extra bit of information denoting the truth value of the formula φ at that position. Under this view, the composition of formulas corresponds to applying the transducers in sequence: the first transducer reads the input, and adds some extra labelling on it. Then a second transducer reads the resulting word, and processes it in a similar way, etc... If we push this view further, we can establish exact correspondences between the class BMA, and suitable cascades of transducers. Furthermore, the comp-height of the formula matches the number of transducers involved in the cascade.

First we introduce some notation. Given a data word $w = (a_1, d_1) \cdots (a_n, d_n)$ the *string projection* of w, denoted by $str(w)$, is the word $a_1 \cdots a_n$. For a class $S = \{i_1, \dots, i_m\}$ the *class projection* corresponding to S, denoted as $str(w|_S)$, is the finite word $a_{i_1} \cdots a_{i_m}$. For a word $u = b_1 \cdots b_n$, the relabelling of w by u is the data word $(b_1, d_1) \dots (b_n, d_n)$. Similarly the relabelling of the class S in w by $b_1 \cdots b_m$ is the data word $(a'_1, d_1) \cdots (a'_n, d_n)$ where $a'_i = b_j$ if $i = i_j$ and a_i otherwise.

The *marking* of a position i in the data word w, in notation $m(i)$, is the subset of zeroary modalities Z satisfied by i. The *marked string projection* of w, denoted as $\mathrm{mstr}(w)$, is the word $(a_1, m(1)) \cdots (a_n, m(n))$ over the alphabet $\Sigma \times 2^Z$. For a class $S = \{i_1, \dots, i_n\}$ the *marked class projection* of S is the finite word $(a_{i_1}, m(i_1)) \cdots (a_{i_n}, m(i_n))$, and it is denoted as $\mathrm{mstr}(w|_S)$.

A *functional letter-to-letter transducer* $\mathcal{A} : \Sigma^* \to \Gamma^*$ over words is a nondeterministic finite state letter-to-letter transducer such that every input word $w \in \Sigma^*$ has at most one output word $\mathcal{A}(w) \in \Gamma^*$.

We next disclose two forms of transductions possible by a word transducer on data words. Let $\mathcal{A} : (\Sigma \times 2^Z)^* \to \Gamma^*$ be a functional letter-to-letter transducer.

The automaton $\mathcal{A}$ acts as a *global transducer* when it runs on the marked string projection $\mathrm{mstr}(w)$ of the input data word $w \in (\Sigma \times \mathcal{D})^*$. If the run succeeds then the unique output data word $w' \in (\Gamma \times \mathcal{D})^* = \mathcal{A}(w)$ (by abuse of notation) is the relabelling of w with the word $\mathcal{A}(\mathrm{mstr}(w))$.

Automaton $\mathcal{A}$ is a *class transducer* when for each class S in the input data word w, a copy of the automaton $\mathcal{A}$ runs on the marked class projection $\mathrm{mstr}(w|_S)$. If all the runs succeed then the unique output data word $\mathcal{A}(w)$ (by abuse of notation) is the relabelling of each class of S of w by $\mathcal{A}(\mathrm{mstr}(w|_S))$.

▶ **Definition 6.** A *cascade of class and global transducers over data words* (hereafter simply cascade) $\mathcal{C}$ is a sequence $\langle \Sigma = \Sigma_0, \mathcal{A}_1, \Sigma_1, \ldots, \Sigma_{n-1}, \mathcal{A}_n, \Sigma_n \rangle$ such that $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is a sequence of class and global transducers over data words and for each i, the transducer $\mathcal{A}_i$ has input alphabet $\Sigma_{i-1} \times 2^Z$ and output alphabet Σ_i. Sets Σ_0, Σ_n are respectively the *input* and *output* alphabets of the cascade $\mathcal{C}$ and n is the *height* of the cascade.

The cascade $\mathcal{C}$ has a *successful run* on a given data word w if there is a sequence of data words $w_0 = w, w_1, \ldots, w_{n-1}, w_n$ such that each transducer $\mathcal{A}_i$ has a successful run on w_{i-1} outputing the data word w_i. The data word w_n is the output of the cascade $\mathcal{C}$, in notation $\mathcal{C}(w) = w_n$. The language accepted by the cascade $\mathcal{C}$, denoted as $L(\mathcal{C})$, is the set of all data words w on which $\mathcal{C}$ has a successful run.

Two cascades $\mathcal{C}_1$ and $\mathcal{C}_2$ can be composed to form the cascade $\mathcal{C}_1 \circ \mathcal{C}_2$ if the output alphabet of $\mathcal{C}_1$ and the input alphabet of $\mathcal{C}_2$ are the same. Composition of cascades is the natural analogue of composition of formulas; this is expressed by the following proposition.

▶ **Proposition 7.** *Let L be a set of data words. Then the following statements are equivalent.*

1. *L is definable by a formula in* BMA *of comp-height k.*
2. *L is recognisable by a cascade of height k.*

4 Separation of the fragments BMA and BR

In this section we prove the main theorem of the paper, namely the separation of the fragments of BMA and BR. More precisely it is shown that there is a formula in BR that has no equivalent formula in BMA. We start by presenting our technical tool, namely combinatorial expressions [7].

4.1 Combinatorial expressions

Put simply, combinatorial expressions are *circuits* over a *data domain* $\mathcal{E}$. For our purposes it is sufficient to assume that $\mathcal{E}$ is a set that contains all the usual data types such as Booleans, integers, finite words etc. We form expressions by composing variables (denoted by X, Y etc.) and functions (denoted by f, g etc.) whose domains and ranges are explicitly specified. A variable X has *range* $E \subseteq \mathcal{E}$, denoted as $X : E$, if it takes values from the set E. We say a function $f : E_1 \times \cdots \times E_k \to F$, where $E_1, \ldots, E_k, F \subseteq \mathcal{E}$, has *arity* k, *domain* $E_1 \times \cdots \times E_k$ and range F. The expressions are built using two specific classes of functions (called gates), namely:

- *binary functions* — when $k \leq 2$, and,
- *finitary functions* — when each of $E_1, \ldots, E_k$ is finite.

For example the addition on integers $+ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is a binary function, whereas the Boolean disjunction of k inputs $\vee : \{0,1\}^k \to \{0,1\}$ is a finitary function.

▶ **Definition 8.** *Combinatorial expressions* are defined inductively;
- a variable $X : E$ is a combinatorial expression with range E, and *depth* 0.
- if $f : E_1 \times \cdots \times E_k \to F$ is a binary or a finitary function, and $t_1, \ldots, t_k$ are combinatorial expressions with ranges $E_1, \ldots, E_k$ and depths $d_1, \ldots, d_k$ respectively, then $f(t_1, \ldots, t_k)$ is a combinatorial expression with range F and depth $\max(d_1, \ldots, d_k) + 1$.

Let $t(\bar{X})$ be a combinatorial expression that contains (possibly vacuously) the variables $\bar{X} = X_1 : E_1, \ldots, X_n : E_n$. For the valuation $\bar{a} = a_1, \ldots, a_n$, where $a_i \in E_i$ for each i, of the variables $\bar{X}$, the value of the expression t, denoted as $t(\bar{a})$, is defined inductively; if t is a variable X_i then $t(\bar{a}) = a_i$, and if $t = f(t_1, \ldots, t_k)$ then $t(\bar{a}) = f(t_1(\bar{a}), \ldots, t_k(\bar{a}))$. Assume $F \subseteq \mathcal{E}$ is the range of the expression t. Naturally t defines a map $[\![t]\!] : \bar{a} \to t(\bar{a})$ from the set $E_1 \times \cdots \times E_n$ to the set F. Given a map $m : E_1 \times \cdots \times E_n \to F$, where $E_1, \ldots, E_n, F \subseteq \mathcal{E}$, we say the map is *recognised* by an expression t if $[\![t]\!] = m$. A particular case is when the range of the map m is restricted to a set of size two (without loss of generality $\{0, 1\}$); in which case we say that t recognises the *property* $\{a_1, \ldots, a_n : m(a_1, \ldots, a_n) = 1\}$.

▶ **Example 9.** Each map $f : E^n \to F$, for some $E, F \subseteq \mathcal{E}, n \in \mathbb{N}$, has an expression of depth $\lceil \log n \rceil + 1$ recognising it. Let $cat : E^* \times E^* \to E^*$ be the concatenation operation on words over the alphabet E and let $t(X_1 : E, \ldots, X_n : E)$ be an expression of depth $\lceil \log n \rceil$ that consists of only the function cat and that computes the concatenation of the inputs. Let $u : E^* \to F$ be a binary function on words over E such that $u(e_1 \cdots e_n) = f(e_1, \ldots, e_n)$. The map f is recognised by expression $u(t(X_1 : E, \ldots, X_n : E))$.

▶ **Example 10.** Consider the set P_n of n-tuples $(u_1, \ldots, u_n)$ of words in $\{0, 1\}^*$ that all have equal length. The property P_n is recognised by the expression

$$t = \bigwedge (el(X_1, X_2), \ldots, el(X_1, X_n), el(X_2, X_3), \ldots, el(X_2, X_n), \ldots, el(X_{n-1}, X_n))$$

where $\bigwedge$ is the Boolean conjunction on $n \cdot (n-1)/2$ inputs and $el : A^* \times A^* \to \{0, 1\}$ is the function on words defined as $el(u, v) = 1$ iff the words u and v are of the same length. The function $\bigwedge$ is finitary and the function el is binary. The expression t has depth 2.

In the previous example, regardless of the value of n the expression t has a constant depth. But there exists properties for which it is not the case.

▶ **Definition 11.** Let V_n be the set of n-tuples $(u_1, \ldots, u_n)$ of words over the alphabet $\{0, 1\}$ such that:
1. the words $u_1, \ldots, u_n$ are of the same length, and;
2. there exists a position $1 \leq i \leq |u_1|$ such that the ith letter of each of u_1 to u_n is 1.

It is shown in [7] that,

▶ **Theorem 12.** *There is no expression of depth at most k that recognises the property V_{2^k+1}.*

4.2 Separation results

We now apply the above theorem to derive our inexpressibility results. The idea is to define a data language B_n that corresponds to the property V_n and to show that if there is a BMA-formula of comp-height k recognising B_n then there is a combinatorial expression of depth $\mathcal{O}(k)$ (precise bound disclosed later) recognising the property V_n. This claim along with the Theorem 12 implies a lower bound on the comp-height of formulas defining the language B_n.

For the proof we rely on data words with a special structure that encode a sequence of words. Let $v_1, \dots, v_n \in \Sigma^*$ be words of identical and even length, say $2\ell \in \mathbb{N}$. A data word $w \in (\Sigma \times \mathcal{D})^*$ is a *coding* of the words $v_1, \dots, v_n \in \Sigma^*$, denoted as $w = \mathit{coding}(v_1, \dots, v_n)$, if $w = w_1 \cdots w_n$ with $v_1 = \mathit{str}(w_1), \dots, v_n = \mathit{str}(w_n)$ and the class relation is the set of tuples $(k \cdot 2\ell + 2i, (k+1) \cdot 2\ell + 2i - 1)$ for $0 \leq k < n - 1, 1 \leq i \leq \ell$; the position $k \cdot 2\ell + 2i$ is the ith even position in the block w_{k+1} and $(k+1) \cdot 2\ell + 2i - 1$ is the ith odd position in the block w_{k+2}. Coding is only defined for words of identical even length and hereafter whenever we say $\mathit{coding}(v_1, \dots, v_n)$ it is understood that $v_1, \dots, v_n$ are of identical even length.

A data word w is a n-coding (or simply a coding when the value n is clear from the context) if it is the coding of some words $v_1, \dots, v_n \in \Sigma^*$. We write n-Codings for the set of all n-codings.

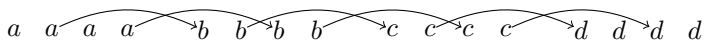


Figure 3 The coding of the words $aaaa, bbbb, cccc, dddd \in \{a, b, c, d\}^*$.

Next we introduce some gates and expressions that we use in the proofs. If w is the coding of $u_1, \dots, u_n \in \Sigma^*$ then $\mathrm{mstr}(w) = m_1(u_1) \cdot m_2(u_2) \cdots m_2(u_{n-1}) \cdot m_3(u_n)$ for binary gates $m_1, m_2, m_3 : \Sigma^* \to (\Sigma \times 2^Z)^*$ such that:

1. For or all words $u = a_1 \cdots a_{2\ell} \in \Sigma^*, 2\ell > 2$

$$m_1(u) = (a_1, x_1) \cdots (a_{2\ell}, x_{2\ell}) \text{ where } x_i = \begin{cases} \{\mathit{fst}^g, \mathit{fst}^c, \mathit{lst}^c\} & \text{if } i = 1, \\ \{\mathit{fst}^c, \mathit{lst}^c\} & \text{if } i \text{ is odd and } i \neq 1, \\ \{\mathit{fst}^c\} & \text{if } i \text{ is even.} \end{cases}$$

$$m_2(u) = (a_1, x_1) \cdots (a_{2\ell}, x_{2\ell}) \text{ where } x_i = \begin{cases} \{\mathit{lst}^c\} & \text{if } i \text{ is odd,} \\ \{\mathit{fst}^c\} & \text{if } i \text{ is even.} \end{cases}$$

$$m_3(u) = (a_1, x_1) \cdots (a_{2\ell}, x_{2\ell}) \text{ where } x_i = \begin{cases} \{\mathit{lst}^c\} & \text{if } i \text{ is odd,} \\ \{\mathit{fst}^c, \mathit{lst}^c\} & \text{if } i \text{ is even and } i \neq 2\ell, \\ \{\mathit{fst}^c, \mathit{lst}^c, \mathit{lst}^g\} & \text{if } i = 2\ell. \end{cases}$$

2. For each word $ab \in \Sigma^2$,

$$m_1(ab) = (a, \{\mathit{fst}^c, \mathit{fst}^g, \mathit{lst}^c\})(b, \{\mathit{fst}^c, \mathsf{S}\}) \ , \qquad m_2(ab) = (a, \{\mathit{lst}^c, \mathsf{P}\})(b, \{\mathit{fst}^c, \mathsf{S}\}) \ ,$$
$$m_3(ab) = (a, \{\mathit{lst}^c, \mathsf{P}\})(b, \{\mathit{fst}^c, \mathit{lst}^c, \mathit{lst}^g\}) \ .$$

3. For words of odd length the functions m_1, m_2, m_3 are fixed arbitrarily.

Let $\mathit{is}\epsilon : \Sigma^* \to \{0, 1\}$ be the binary gate defined as $\mathit{is}\epsilon(w) = 1$ precisely when $w \in \Sigma^*$ is not the empty word. Let $bI : \Sigma^* \times \{0, 1\} \to \Sigma^*$ be the binary function $bI(x, 1) = x$ and $bI(x, 0) = \epsilon$. For variables $\bar{X} = X_1 : \Sigma^*, \dots, X_n : \Sigma^*$, let $NE(\bar{X})$ be the expression $\bigwedge(\mathit{is}\epsilon(X_1), \dots, \mathit{is}\epsilon(X_n))$ of depth 2 that recognises the property that none of the input words is the empty word. Sometimes we use these gates and expressions over other alphabets, and then it is understood that the domains of the functions are appropriately defined.

Next we prove that class transductions and global transductions on n-codings can be defined by expressions of fixed height (irrespective of n). To summarise the intuition, let $w = w_1 \cdots w_n$ be the coding of the words $u_1, \dots, u_n \in \Sigma^*$ such that $\mathit{str}(w_i) = u_i$. Assume $\mathcal{A} : (\Sigma \times 2^Z)^* \to \Gamma^*$ is a class transducer that has a successful run on w and let $\mathcal{A}(w) = w' = w'_1 \cdots w'_n \in (\Gamma \times \mathcal{D})^*$ where w'_i is a relabelling of w_i. Observe that the only

other positions in the class of a position in w_i appear either in w_{i-1} or w_{i+1}. Therefore to compute $str(w'_i)$ it suffices to know the words u_{i-1}, u_i, u_{i+1} and hence there is an expression that takes as inputs u_{i-1}, u_i, u_{i+1} and outputs the word $str(w'_i)$.

▶ **Lemma 13.** *For each class transducer $\mathcal{A} : (\Sigma \times 2^Z)^* \to \Gamma^*$ and each $n \in \mathbb{N}$ there exist combinatorial expressions $e_1(\bar{X}), \ldots, e_n(\bar{X})$, where $\bar{X} = X_1 : \Sigma^*, \ldots, X_n : \Sigma^*$, of depth 7 such that for all n-tuple $\bar{u} = (u_1, \ldots, u_n)$ of words in Σ^* of identical even length $coding(e_1(\bar{u}), \ldots, e_n(\bar{u})) = \mathcal{A}(coding(\bar{u}))$.*

Next we prove a similar claim for global transducers. The idea is as follows. Assume $\mathcal{A} : (\Sigma \times 2^Z)^* \to \Gamma^*$ is a global transducer and let $w = w_1 \cdots w_n$ be the coding of the words $u_1, \ldots, u_n \in \Sigma^*$ such that $str(w_i) = u_i$. Assume that $\mathcal{A}$ has a successful run on w and let $\mathcal{A}(w) = w' = w'_1 \cdots w'_n \in (\Gamma \times \mathcal{D})^*$ where w'_i is a relabelling of w_i. To compute $str(w'_i)$ it suffices to know the word u_i and the pair (p, q) of states of the automaton $\mathcal{A}$ which are respectively the state of the automaton $\mathcal{A}$ before and after reading the word $\mathrm{mstr}(u_i)$ on the unique run on $\mathrm{mstr}(w)$. Among these, the pair (p, q) can be computed a finitary function that aggregates the set of all possible partial runs of $\mathcal{A}$ on each of the words $u_1, \ldots, u_n$ and hence an expression of fixed height can compute the word $str(w'_i)$.

▶ **Lemma 14.** *For each global transducer $\mathcal{A} : (\Sigma \times 2^Z)^* \to \Gamma^*$ and each $n \in \mathbb{N}$ there exist combinatorial expressions $e_1(\bar{X}), \ldots, e_n(\bar{X})$, where $\bar{X} = X_1 : \Sigma^*, \ldots, X_n : \Sigma^*$, of depth 5 such that for all n-tuple $\bar{u} = (u_1, \ldots, u_n)$ of words in Σ^* of identical even length $coding(e_1(\bar{u}), \ldots, e_n(\bar{u})) = \mathcal{A}(coding(\bar{u}))$.*

The above two lemmas can be generalised to a similar claim on cascades by induction (on the height of the cascade).

▶ **Lemma 15.** *For a cascade $\mathcal{C} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle$ with input alphabet Σ, and each $n \in \mathbb{N}$ there exist combinatorial expressions $e_1(\bar{X}), \ldots, e_n(\bar{X})$, where $\bar{X} = X_1 : \Sigma^*, \ldots, X_n : \Sigma^*$, of depth at most $7k$ such that for all n-tuple $\bar{u} = (u_1, \ldots, u_n)$ of words in Σ^* of identical even length $coding(e_1(\bar{u}), \ldots, e_n(\bar{u})) = \mathcal{C}(coding(\bar{u}))$.*

Next we define a data language B_n that corresponds to the property V_n.

For a word $w = a_1 a_2 \ldots a_l \in \{0,1\}^*$ we let $pad(w) = 1a_1 1a_2 \cdots 1a_l$. We will also use *pad* as a binary gate. A *bridge* in a data word w is a sequence of positions along a path that consists of alternating class successor and global successor edges. Formally the sequence of positions $i_1, \ldots, i_n$ forms a bridge in w if there exists a sequence of successor and class successor edges $e_1, \ldots e_{n-1}$ in w such that for each $1 \leq j < n$, $e_j = (i_j, i_{j+1})$ and for each $1 \leq j < n-1$, e_j is a successor edge iff e_{j+1} is a class successor edge. A bridge is a-labelled, for $a \in \Sigma$, if all the positions in the bridge are labelled by the letter a.

▶ **Definition 16.** Let $B_n \subseteq (\{0,1\} \times \mathcal{D})^*$ be the set of all data words w such that w has a 1-labelled bridge $i_1, \ldots, i_{2n-1}$ (connected by a path of $2n-2$ edges), and

1. all positions to the left of i_1 are first positions of classes,
2. all positions to the right of i_{2n-1} are last positions of classes, and
3. the path corresponding to the bridge starts with a class successor edge.

Define the data language $B = \bigcup_{n=1}^{\infty} B_n$.

The language B_n is defined by the BMA formula (also in unary-DataLTL) of comp-height $2n+1$,

$$\mathtt{F}^g\left(\mathtt{H}^g fst^c \wedge (\mathtt{1X}^c\mathtt{1X}^g)^n \mathtt{G}^g lst^c\right) \qquad \text{where } \mathtt{1X}^g\varphi \text{ stands for the formula } (1 \wedge \mathtt{X}^g\varphi), \text{ and } \mathtt{1X}^c\varphi \text{ for } (1 \wedge \mathtt{X}^c\varphi). \tag{1}$$

Similarly the language B is defined by the BR formula

$$fst^c\, \mathtt{U}^g\, (\mu x.(\mathtt{1X}^c\mathtt{1X}^g x \vee \mathtt{1X}^c\mathtt{1X}^g\mathtt{G}^g lst^c)) \ . \tag{2}$$

▶ **Proposition 17.** *Let $(u_1, \ldots, u_n)$ be a tuple of words of identical length over the alphabet $\{0,1\}$. Then the following are equivalent.*

1. $(u_1, \ldots, u_n) \in V_n$.
2. *The data word $w = coding(pad(u_1), \ldots, pad(u_n))$ is in the language B_n.*

Figure 4 The data word w corresponding to the words $a_1a_2, b_1b_2, c_1c_2, d_1d_2$, and a bridge of length 7 in w.

For a data language $L \subseteq (\Sigma \times \mathcal{D})^*$ we write $L^c = \{w \in (\Sigma \times \mathcal{D})^* \mid w \notin L\}$ for the complement of L. The data language $L \subseteq (\Sigma \times \mathcal{D})^*$ *separates* the data languages $L_1, L_2 \subseteq (\Sigma \times \mathcal{D})^*$ if $L_i \cap L = \emptyset$ and $L_{1-i} \subseteq L$ for some $i \in \{0,1\}$. A cascade $\mathcal{C}$ (respectively a formula φ) separates the data languages L_1, L_2 if $L(\mathcal{C})$ (respectively $L(\varphi)$) separates L_1, L_2.

▶ **Lemma 18.** *If there is a cascade $\mathcal{C}$ of height k that separates the data languages $L_1 = B_n \cap n$-Codings, $L_2 = (B_n)^c \cap n$-Codings then there is a combinatorial expression of depth $7k+4$ recognising the property V_n.*

Proof. Assume that $\mathcal{C}$ is a cascade of height k separating the languages L_1, L_2. Since cascades (of height k) are closed under complementation, without loss of generality assume that $L(\mathcal{C}) \supseteq L_1$ and $L(\mathcal{C}) \cap L_2 = \emptyset$. Therefore the cascade $\mathcal{C}$ produces an output on a data word n-Codings $\ni w \in (\{0,1\} \times \mathcal{D})^*$ if and only if w is in the language B_n. Let $e_1(\bar{X}), \ldots, e_n(\bar{X})$, for $\bar{X} = X_1 : \{0,1\}^*, \ldots, X_n : \{0,1\}^*$, be the combinatorial expressions of depth at most $7k$, guaranteed by the Lemma 15 such that for all n-tuple $\bar{u} = (u_1, \ldots, u_n)$ of words in $\{0,1\}^*$ of identical even length, $coding(e_1(\bar{u}), \ldots, e_n(\bar{u})) = \mathcal{C}(coding(\bar{u}))$.

Let $pad(\bar{X})$ stand for the vector of expressions $pad(X_1), \ldots, pad(X_n)$. We claim that *the expression*

$$e = \bigwedge(NE(e_1(pad(\bar{X})), \ldots, e_n(pad(\bar{X})), t(X_1, \ldots, X_n)) \ ,$$

where t is the expression from Example 10 for the alphabet $\{0,1\}$ that checks if all the input words are of the same length, computes the property V_n. The expression e has depth at most $7k+4$. To show the claim it is enough to verify that for a tuple $\bar{u} = (u_1, \ldots, u_n)$ of words from $\{0,1\}^*$ of equal length, none of the words $v_1 = e_1(pad(\bar{u})), \ldots, v_n = e_n(pad(\bar{u}))$ is the empty word if and only if $\bar{u} \in V_n$. By Lemma 15, the words v_1 to v_n are nonempty iff $\mathcal{C}$ accepts the data word $w = coding(pad(\bar{u}))$. By assumption, the data word w is accepted by the cascade $\mathcal{C}$ iff $w \in B_n$. By Lemma 17, the data word w is in the language B_n iff $\bar{u}$ is in the property V_n. Hence the claim is proved. ◀

We are now ready for the main theorem;

▶ **Theorem 19** (Separation). *Let $N = 7k+4$.*

1. *The data languages $L_1 = B_{2^N+1} \cap (2^N+1)$-Codings and $L_2 = (B_{2^N+1})^c \cap (2^N+1)$-Codings are not separable by a formula in* BMA *of comp-height k.*
2. *The data language B_{2^N+1} is not definable by a formula in* BMA *of comp-height k.*
3. *Class of* BMA *definable languages form a hierarchy under composition height; more precisely for every k there exists a* BMA*-formula φ with comp-height k that has no equivalent formula of comp-height $k-1$.*
4. *The class of* BMA *definable languages is strictly subsumed by the class of* BR *definable languages.*

Proof.

1. Proof by contradiction. Assume that the data languages L_1, L_2 are separable by a BMA formula φ of comp-height k. This implies that there is cascade of height k separating L_1, L_2. By Lemma 18 there is an expression of depth N recognising the property V_{2^N+1}. This is in contradiction with Theorem 12.
2. Follows from (**1**).
3. From (**2**) and the Equation (1), B_{2^N+1} is definable by a BMA formula of comp-height $2\cdot(2^N+1)+1$ but not by any formula of comp-height k. Therefore (†) the set of languages defined by BMA^k is strictly contained in the set of languages defined by $\mathrm{BMA}^{2\cdot(2^N+1)+1}$. It only remains to separate the languages definable by BMA^k and the languages definable by BMA^{k+1}, for all k. We prove this claim by contradiction. Assume that ($\star$) there is some $m \in \mathbb{N}$ such that for every formula in BMA^{m+1} there is an equivalent formula in BMA^m. We claim that $f$or every formula in BMA^{m+2} there is an equivalent formula in BMA^m as well. To prove the claim, let $\chi = \psi(\varphi_1, \ldots, \varphi_n)$ be an arbitrary formula in BMA^{m+2} such that $\psi \in \mathrm{BMA}^1$ and $\varphi_1, \ldots, \varphi_n \in \mathrm{BMA}^{m+1}$. By assumption ($\star$) there exist formulas $\varphi'_1, \ldots, \varphi'_n \in \mathrm{BMA}^m$ equivalent to the formulas $\varphi_1, \ldots, \varphi_n$ respectively. Therefore the formula $\chi' = \psi(\varphi'_1, \ldots, \varphi'_n)$ is equivalent to the formula χ and is in BMA^{m+1}. Applying the assumption ($\star$) again there is a formula $\chi'' \in \mathrm{BMA}^m$ equivalent to χ' and hence also to χ, and hence the claim is proved. Extending this argument, by induction on k, it can be shown that for every formula in BMA^{m+k} there is an equivalent formula in BMA^m. This is in contradiction with the statement (†). Hence the statement is proved.
4. We claim that the data language B is not definable by any BMA formula. For the sake of contradiction, assume that there is a BMA formula φ of comp-height k recognising the language B and let $\mathcal{C}$ be the cascade of height k corresponding to φ. We claim that the cascade $\mathcal{C}$ separates the languages L_1 and L_2. Clearly, by definition of the language B, $L_1 \subseteq B$. We need to show that $L_2 \cap B = \emptyset$ and it suffices to prove that for every $w \in (2^N+1)$-Codings if $w \in B$ then $w \notin (B_{2^N+1})^c$. Since any coding w in (2^N+1)-Codings either belongs to B_{2^N+1} or does not belong to B, it follows that if $w \in B$ then $w \notin (B_{2^N+1})^c$. Therefore the cascade $\mathcal{C}$ separates the languages L_1 and L_2 which contradicts (**1**) and hence the claim follows. On the other hand, since B is definable by a formula in BR (Equation 2), the statement is proved. ◀

5 Conclusion

In this paper we studied the some fragments of μ-calculus over data words. We disclosed two fragments that are: the Bounded Reversal fragment (BR) and the Bounded Mode Alternation fragment (BMA) and proved they are separate. BR and BMA happen to form Boolean algebras making them very natural, and relatively expressive logics over data words. We also establish the relationship with earlier logics like FO^2 or DataLTL.

References

1 Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.

2 M. Bojańczyk and S. Lasota. An extension of data automata that captures xpath. In *Logic in Computer Science (LICS), 2010*, pages 243–252, July 2010.

3 Mikołaj Bojańczyk. Data monoids. In *STACS*, pages 105–116, 2011.

4 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

5 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. On the use of guards for logics with data. In *MFCS*, volume 6907 of *LNCS*, pages 243–255. Springer, 2011.

6 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *FSTTCS 2014*, volume 29 of *LIPIcs*, pages 267–278, 2014.

7 Thomas Colcombet and Amaldev Manuel. Combinatorial expressions and lower bounds. In *STACS 2015*, volume 30 of *LIPIcs*, pages 249–261, 2015.

8 S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *Logic in Computer Science (LICS), 2013*, pages 33–42, June 2013.

9 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), April 2009.

10 D. Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science*, 8(1), 2012.

11 D. Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic*, 13(4), 2012.

12 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *Language and Automata Theory and Applications*, pages 561–572. Springer, 2010.

13 M. Jurdziński and R. Lazic. Alternating automata on data trees and xpath satisfiability. *ACM Trans. Comput. Log.*, 12(3):19, 2011.

14 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

15 Ahmet Kara, Thomas Schwentick, and Thomas Zeume. Temporal logics on words with multiple data values. In *FSTTCS*, volume 8 of *LIPIcs*, pages 481–492, 2010.

16 L. Libkin and D. Vrgoc. Regular expressions for data words. In *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 274–288. SPRINGER, 2012.

17 A. Manuel, A. Muscholl, and G. Puppis. Walking on data words. In *Computer Science Theory and Applications*, volume 7913 of *LNCS*, pages 64–75. Springer, 2013.

18 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

Efficient Algorithms for Morphisms over Omega-Regular Languages*

Lukas Fleischer and Manfred Kufleitner

FMI, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{fleischer,kufleitner}@fmi.uni-stuttgart.de

Abstract

Morphisms to finite semigroups can be used for recognizing omega-regular languages. The so-called strongly recognizing morphisms can be seen as a deterministic computation model which provides minimal objects (known as the syntactic morphism) and a trivial complementation procedure. We give a quadratic-time algorithm for computing the syntactic morphism from any given strongly recognizing morphism, thereby showing that minimization is easy as well. In addition, we give algorithms for efficiently solving various decision problems for weakly recognizing morphisms. Weakly recognizing morphism are often smaller than their strongly recognizing counterparts. Finally, we describe the language operations needed for converting formulas in monadic second-order logic (MSO) into strongly recognizing morphisms, and we give some experimental results.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Büchi automata, omega-regular language, syntactic semigroup, recognizing morphism, MSO

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.112

1 Introduction

Automata over finite words have a huge number of effective closure properties. Moreover, many problems such as minimization or equivalence of deterministic automata admit very efficient algorithms [6, 7]. The situation over infinite words is quite similar, but with the major difference that many operations are less efficient. There are many different automaton models for accepting languages of infinite words, the so-called ω-regular languages. Each of these models has its advantages and disadvantages. For instance, deterministic Büchi automata are less powerful than nondeterministic Büchi automata [15]. And only very few automaton models admit efficient minimization algorithms; for example, minimization of deterministic finite automata can be applied to the lasso automata in [2].

The theory of finite semigroups and automata is tightly connected [11]. Since the semigroup for a language can be exponentially bigger than its automaton, semigroups have very rarely been considered in the context of efficient algorithms. There is also an algebraic approach to ω-regular languages by using morphisms to finite semigroups, see e.g. [9, 15]. Among the many nice properties of this approach are minimal morphisms — the so-called syntactic morphisms — and easy complementation. As for finite words, the semigroup for an ω-regular language can be exponentially bigger than its Büchi automaton. However, since many operations for ω-regular languages are less efficient than for regular languages over

* This work was supported by the DFG grants DI 435/5-2 and KU 2716/1-1.

finite words, the drawback of this exponential blow-up in size is less serious. This is even more so when minimizing all intermediate objects.

A typical algorithm for computing the syntactic morphism of a regular language over finite words is to minimize the (deterministic) automaton defined by the Cayley graph of a morphism, and then the syntactic morphism is given by the transition semigroup of the minimal automaton. This approach does not work for infinite words and we therefore give a direct algorithm for computing the syntactic morphism. Our algorithm is an adaptation of Hopcroft's minimization algorithm [6] and its running time is quadratic in the size of the semigroup. We show that this is rather optimal.

There are two different modes for recognizing omega-regular languages by a morphism to a finite semigroup: *weak* and *strong* recognition. Strong recognition is a special case of weak recognition. Easy complementation and the computation of the syntactic morphism only works for strong recognition. We show how to test whether a given weak recognition is actually strong. Another useful tool for morphisms is the computation of the so-called conjugacy classes.

As an application, we consider the translation of MSO formulas into strongly recognizing morphisms. To this end, we show that a powerset construction preserves strong recognition, and that this construction can be used for computing the image under a length-preserving morphism. Finally, we give the test results of some translations from MSO to strong recognition. Deciding the satisfiability of an MSO formula is non-elementary [13] and therefore, minimization of intermediate objects is usually very helpful for solving some special cases. This is confirmed by our test results.

2 Preliminaries

Words. Let A be a finite *alphabet*. The elements of A are called *letters*. A *finite word* is a sequence $a_1a_2\cdots a_n$ of letters of A and an *infinite word* is an infinite sequence $a_1a_2\cdots$. The empty word is denoted by ε. The set of non-empty finite words over A is A^+. Let K be a set of finite words and let L be a set of infinite words. We set $KL = \{u\alpha \mid u \in K, \alpha \in L\}$, $K^+ = \{u_1u_2\cdots u_n \mid n \geqslant 1, u_i \in K\}$ and $K^* = K^+ \cup \{\varepsilon\}$. Moreover, if $\varepsilon \notin K$ we define the *infinite iteration* $K^\omega = \{u_1u_2\cdots \mid u_i \in K\}$. A natural extension to $K \subseteq A^*$ is $K^\omega = (K \setminus \{\varepsilon\})^\omega$.

Finite semigroups. Let S be a finite semigroup. An element e of S is *idempotent* if $e^2 = e$. The set of idempotent elements of S is denoted by $E(S) = \{e \in S \mid e^2 = e\}$. For each $s \in S$ the set $\{s^k \mid k \geqslant 1\}$ of all powers of s is finite and it contains exactly one idempotent element.

A semigroup S is called *X-generated* if X is a subset of S and every element of S can be written as a product of elements of X. The *right Cayley graph* of an X-generated semigroup S has S as vertices and its labeled edges are the triples of the form (s, a, sa) for $s \in S$ and $a \in X$. The *left Cayley graph* of S is defined analogously with edges of the form (s, a, as). The definitions of Cayley graphs depend on the choice of the set X. In the following, when a surjective morphism $h\colon A^+ \to S$ is given, we choose $X = h(A)$ as the set of generators.

Green's relations are an important tool in the study of finite semigroups. We denote by S^1 the monoid that is obtained by adding a new neutral element 1 to S. For $s, t \in S$ let

$$s \mathrel{\mathcal{R}} t \text{ if there exist } q, q' \in S^1 \text{ such that } sq = t \text{ and } tq' = s,$$
$$s \mathrel{\mathcal{L}} t \text{ if there exist } p, p' \in S^1 \text{ such that } ps = t \text{ and } p't = s.$$

These relations are equivalence relations and the equivalence classes of $\mathcal{R}$ (resp. $\mathcal{L}$) are called *$\mathcal{R}$-classes* (resp. *$\mathcal{L}$-classes*). The $\mathcal{R}$-classes (resp. $\mathcal{L}$-classes) of a semigroup S can be

computed in time linear in $|S|$ by applying Tarjan's algorithm to the right (resp. left) Cayley graph of S, see [5].

An element $(s, e) \in S \times E(S)$ is a *linked pair* if $se = s$. Two linked pairs (s, e) and (t, f) are *conjugate*, written as $(s, e) \sim (t, f)$, if there exist $x, y \in S$ such that $sx = t$, $xy = e$ and $yx = f$. The conjugacy relation $\sim$ on the set of linked pairs is an equivalence relation, see e.g. [9]. The equivalence classes of $\sim$ are called *conjugacy classes*. A set P of linked pairs is *closed under conjugation* if it is a union of conjugacy classes.

Recognition by morphisms. A language $L \subseteq A^\omega$ is *regular* (or *ω-regular*) if it is recognized by some finite Büchi automaton, see e.g. [3]. The family of regular languages is closed under Boolean operations, i.e., set union, set intersection and complementation. We now describe algebraic recognition modes for regular languages. Let $h\colon A^+ \to S$ be a morphism onto a finite semigroup S. For $s \in S$, we set $[s] = h^{-1}(s)$ and for $P \subseteq S \times S$, we set

$$[P] = \bigcup_{(s,t)\in P} [s][t]^\omega$$

if h is understood from the context. A language $L \subseteq A^\omega$ is *weakly recognized* by a morphism $h : A^+ \to S$ if there exists a set of linked pairs $P \subseteq S \times E(S)$ with $L = [P]$. If in addition P is closed under conjugation, then h *strongly recognizes* L. Another well-known characterisation of strong recognition is the following, see e.g. [4].

▶ **Proposition 1.** *Let $h : A^+ \to S$ be a morphism onto a finite semigroup. Then h strongly recognizes L if and only if $[s][t]^\omega \cap L \neq \emptyset$ implies $[s][t]^\omega \subseteq L$ for all $s, t \in S$.*

The *syntactic congruence* $\equiv_L$ of a language $L \subseteq A^\omega$ is defined over A^+ as $u \equiv_L v$ if the equivalences

$$(xuy)z^\omega \in L \Leftrightarrow (xvy)z^\omega \in L \text{ and}$$
$$z(xuy)^\omega \in L \Leftrightarrow z(xvy)^\omega \in L$$

hold for all finite words $x, y, z \in A^*$. Our definition is slightly different but equivalent to the syntactic congruence introduced by Arnold [1]. The congruence classes of $\equiv_L$ form the so-called *syntactic semigroup* $A^+/\equiv_L$ and the *syntactic morphism* $h_L\colon A^+ \to A^+/\equiv_L$ is the natural quotient map. If L is regular, the syntactic semigroup of L is finite and h_L strongly recognizes L [1, 9].

Model of computation. Morphisms $h\colon A^+ \to S$ are given implicitly through a mapping $f\colon A \to S$ with $f(a) = h(a)$ for all $a \in A$. We assume that for finite semigroups S, multiplications can be performed in constant time. Some algorithms only perform multiplications of the form $h(a) \cdot s$ or $s \cdot h(a)$ where h is a morphism, s is an element of S and a is a letter. In that case, semigroups can be represented efficiently by their left and right Cayley graphs. For two elements $s, t \in S$ we can check in constant time whether $s = t$ and it is possible to organize elements of S in a hash map such that operations on subsets of S can be implemented efficiently. When a set $P \subseteq S \times S$ is part of the input, we assume that for each $s, t \in S$ one can check in constant time whether $(s, t) \in P$.

3 Conversion between Büchi automata, weak and strong recognition

In this section, we describe well-known constructions for the conversion between the different acceptance modes for regular languages. For details and proofs, we refer to [9, 10, 15].

3.1 From Büchi automata to strong recognition

In the case of finite words, when proving that each regular language is recognizable by a morphism onto a finite semigroup, one usually considers the transition semigroup of a finite automaton. However, when applying the same construction to Büchi automata, the resulting morphism only weakly recognizes the language. In this section, we describe a construction to convert a Büchi automaton $\mathcal{A} = (Q, A, \delta, I, F)$ into a semigroup S and a morphism $h\colon A^+ \to S$ that strongly recognizes $L(\mathcal{A})$.

For states $p, q \in Q$ and a finite word $u \in A^+$, we write $p \xrightarrow{u} q$ if there exists a sequence $q_0 a_1 q_1 a_2 q_2 \cdots q_{n-1} a_n q_n$ with $q_0 = p$, $q_n = q$ and $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for all $i \in \{0, \ldots, n-1\}$. If, additionally, $q_i \in F$ for some $i \in \{0, \ldots, n\}$, we write $p \xrightarrow[F]{u} q$. We now assign to each word $u \in A^+$ a $Q \times Q$ matrix $h(u)$ defined by

$$(h(u))_{pq} = \begin{cases} 1 & \text{if } p \xrightarrow{u} q \text{ but not } p \xrightarrow[F]{u} q \\ 2 & \text{if } p \xrightarrow[F]{u} q \\ 0 & \text{otherwise} \end{cases}$$

A routine verification shows that this naturally extends the image of A^+ under h to a semigroup S. We say that a linked pair (R, E) where $R = (r_{pq})_{p,q \in Q}$ and $E = (e_{pq})_{p,q \in Q}$ is *accepting* if there exist states $p, q \in Q$ such that $r_{pq} \geqslant 1$ and $e_{qq} = 2$. One can now verify that the set P of all accepting linked pairs is closed under conjugation and that $[P] = L(\mathcal{A})$.

3.2 From weak recognition to Büchi automata

Suppose we are given a morphism $h\colon A^+ \to S$ onto a finite semigroup S that weakly recognizes a language L, i.e., $L = [P]$ for some set of linked pairs $P \subseteq S \times E(S)$. One can use the following construction from [10] to obtain a Büchi automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$.

The set of states is $Q = S^1 \times E(S)$, the set of initial states is $I = P$ and the set of final states is $F = \{1\} \times E(S)$. The transition relation δ consists of all tuples of the form $((s, e), a, (t, e)) \in Q \times A \times Q$ where $h(a)t = s$ or $h(a)t = se$.

By combining the constructions from this and the previous subsection, we also obtain a construction to convert a morphism that weakly recognizes a language L into a morphism that strongly recognizes L. There are also direct, more efficient constructions, to perform this conversion, see e.g. [9]. The converse direction is trivial since, by definition, a morphism $h\colon A^+ \to S$ that strongly recognizes a language L also weakly recognizes L.

4 Computing conjugacy classes

When designing an algorithm that takes a set of linked pairs $P \subseteq S \times E(S)$ as input, it is often convenient to assume that P is closed under conjugation. However, this is not always the case in practice: The input set P might be a proper subset of its closure under conjugation Q such that $[P] = [Q]$. In this section, we describe an algorithm to compute the conjugacy classes efficiently. It justifies the assumption that P is always closed under conjugation in the following sections, particularly in Section 6.

As a warm-up, we first describe how to compute the set F of linked pairs. The linked pairs are exactly the pairs of the form (se, e) with $s \in S$ and $e \in E(S)$. Thus, we first check for each element $e \in S$ whether $e^2 = e$. If the outcome of the check is positive, we perform a depth-first search in the left Cayley graph of S, starting at element e. For each element s that is visited, (s, e) is a linked pair. The total running time of this routine is $\mathcal{O}(|S| + |A| \cdot |F|)$.

An equivalence relation $\equiv$ on the set of linked pairs is called *left-stable* if for all $p \in S$ and for linked pairs (s,e), (t,f) with $(s,e) \equiv (t,f)$, we have $(ps,e) \equiv (pt,f)$. We define an equivalence relation $\approx$ on the set of linked pairs by $(s,e) \approx (t,f)$ if and only if $e \mathcal{L} s \mathcal{R} t \mathcal{L} f$ or $(s,e) = (t,f)$. Its relationship to conjugacy is captured in the following Lemma:

▶ **Lemma 2.** *The conjugacy relation $\sim$ is the finest left-stable equivalence relation coarser than $\approx$.*

Proof. It follows directly from the definitions of linked pairs and conjugacy that $\sim$ is left-stable. Let (s,e) and (t,f) be linked pairs with $(s,e) \approx (t,f)$ and $(s,e) \neq (t,f)$. Since $s \mathcal{R} t$, there exist $q, q' \in S^1$ such that $sq = t$ and $tq' = s$. We set $x = eq$ and $y = fq'$. Now, $sx = seq = sq = t$. Moreover, since $s \mathcal{L} e$, there exists $p \in S^1$ with $ps = e$. Thus, we have $xy = eqy = psqy = pty = ptfq' = ptq' = ps = e$. A similar argument can be used to show that $yx = f$. Hence, (s,e) and (t,f) are conjugate, and $\sim$ is indeed coarser than $\approx$.

In order to show that $\sim$ is the finest relation with these properties, we consider an arbitrary left-stable equivalence relation $\simeq$ on the set of linked pairs which is coarser than $\approx$. We show that $(s,e) \sim (t,f)$ implies $(s,e) \simeq (t,f)$. Let $x, y \in S$ such that $sx = t$, $xy = e$ and $yx = f$. Then we have $ex = xyx = xf$ and $xfy = xyxy = e^2 = e$, which shows that $e \mathcal{R} xf$. Furthermore we have $xf \mathcal{L} f$, since $yxf = f^2 = f$. By the definition of $\approx$, this means that $(e,e) \approx (xf,f)$ and since $\approx$ refines $\simeq$, it follows that $(e,e) \simeq (xf,f)$. Left-stability yields $(s,e) = (se,e) \simeq (sxf,f) = (t,f)$. ◀

Since $\mathcal{R}$-classes and $\mathcal{L}$-classes can be computed in time linear in the size of the semigroup, this allows us to efficiently compute the conjugacy classes as shown in Algorithm 1. We use a so-called *disjoint-set data structure* that provides two operations on a partition. $\mathrm{Find}(s,e)$ returns a unique element from the class that contains (s,e), i.e., if (s,e) and (t,f) are in the same class, we have $\mathrm{Find}(s,e) = \mathrm{Find}(t,f)$. $\mathrm{Union}((s,e),(t,f))$ merges the classes of (s,e) and (t,f). To simplify the notation we also introduce an operation $\mathrm{Union}^+(R)$ for subsets R of $S \times S$ that merges all classes with elements in R. $\mathrm{Union}^+(R)$ can be implemented using $|R| - 1$ atomic Union operations. The partition is initialized with singleton sets $\{(s,e)\}$ for all linked pairs (s,e). The second data structure used in the algorithm is a set $T \subseteq 2^F$.

Algorithm 1 Computing conjugacy classes

```
initialize T with the non-trivial equivalence classes of ≈
for all R ∈ T do Union⁺(R) end for
while T ≠ ∅ do
    remove some set R from T
    for all a ∈ A do
        R' ← ∅
        for all (s,e) ∈ R do R' ← R' ∪ {Find(h(a)s, e)} end for
        if |R'| > 1 then
            Union⁺(R')
            T ← T ∪ {R'}
        end if
    end for
end while
```

To prove the correctness and running time of the algorithm, one can combine Lemma 2 with arguments similar to those given in the correctness and running time proofs of the

Hopcroft-Karp equivalence test [7]. We first show that the relation induced by the final partition is left-stable:

▶ **Lemma 3.** *Let (s, e) and (t, f) be linked pairs of the same class upon termination, then, for each $a \in A$, the pairs $(h(a)s, e)$ and $(h(a)t, f)$ are in the same class as well.*

Proof. We write $\mathrm{Find}_i(s, e) = \mathrm{Find}_i(t, f)$ if (s, e) and (t, f) belong to the same class after the i-th iteration of the *while*-loop. The index ∞ is used to describe the situation upon termination.

Let i be minimal such that for some pairs $(s, e), (t, f)$ and a letter $a \in A$, we have $\mathrm{Find}_i(s, e) = \mathrm{Find}_i(t, f)$ and $\mathrm{Find}_\infty(h(a)s, e) \neq \mathrm{Find}_\infty(h(a)t, f)$. Note that $i > 0$ because otherwise, a set containing both (s, e) and (t, f) would be added to T during initialization. Hence, there exists a pair (s', e') with $\mathrm{Find}_{i-1}(s', e') = \mathrm{Find}_{i-1}(s, e)$ and a pair (t', f') with $\mathrm{Find}_{i-1}(t', f') = \mathrm{Find}_{i-1}(t, f)$ such that $\mathrm{Union}^+(R)$ is executed for some set $R \supseteq \{(s', e'), (t', f')\}$. By choice of i, we have $\mathrm{Find}_\infty(h(a)s, e) = \mathrm{Find}_\infty(h(a)s', e')$ and $\mathrm{Find}_\infty(h(a)t, f) = \mathrm{Find}_\infty(h(a)t', f')$. Since we add the set R to T in iteration i, the equality $\mathrm{Find}_\infty(h(a)s', e') = \mathrm{Find}_\infty(h(a)t', f')$ holds as well, and thus $\mathrm{Find}_\infty(h(a)s, e) = \mathrm{Find}_\infty(h(a)t, f)$, a contradiction. ◀

There is of course a dual statement for the pairs $(s \cdot h(a), e)$ and $(t \cdot h(a), f)$.

▶ **Theorem 4.** *Let F be the set of linked pairs of S. When Algorithm 1 terminates, the classes of the partition correspond to the conjugacy classes of F. Furthermore, the algorithm executes at most*

- $|F| - 1$ Union *operations and*
- $2\,|A|\,(|F| - 1)$ Find *operations.*

Proof. By Lemma 3, the relation induced by the final partition is left-stable and throughout the main algorithm, two classes are only merged when required to establish this property. Thus, the relation is the finest left-stable equivalence relation coarser than $\approx$ and, by Lemma 2, equivalent to the conjugacy relation.

The number of Union operations is bounded by $|F| - 1$ since each operation reduces the number of classes in the partitions by 1. Let $R_1, \ldots, R_k$ be the sets that are added to T during the execution of the algorithm. Whenever one of the sets R_i is inserted into T, $|R_i| - 1$ Union operations are executed. Thus, we have

$$\sum_{i=1}^{k} \big(|R_i| - 1\big) \leqslant |F| - 1.$$

When R_i is removed from T, exactly $|A| \cdot |R_i|$ Find operations are executed in the same iteration of the *while*-loop. The total number of Find operations is therefore bounded by

$$\sum_{i=1}^{k} |A| \cdot |R_i| \leqslant \sum_{i=1}^{k} |A| \cdot (2\,|R_i| - 2) \leqslant 2\,|A| \cdot (|F| - 1)$$

where the first inequality follows from the fact that each of the sets R_i contains at least two elements. ◀

A sequence of n Union- and m Find-operations can be performed in $\mathcal{O}(n + m \cdot \alpha(n))$ time where $\alpha(n)$ denotes the extremely slow-growing *inverse Ackermann function* [14]. Thus, when considering a fixed-size alphabet, the total running time of our algorithm is "almost linear" in the number of linked pairs.

5 Testing for strong recognition

Common decision problems, such as the *universality problem* or the *inclusion problem*, are easy in the case of strong recognition. In the context of weak recognition, the algorithm presented in this section is a powerful tool to answer a broad range of similar problems. Given a morphism $h\colon A^+ \to S$ onto a finite semigroup S and two sets of linked pairs $P, Q \subseteq S \times E(S)$, it can be used to check whether $[P] \subseteq [Q]$. In particular, it allows for testing whether the morphism strongly recognizes a language $L = [P]$ by first computing the closure Q of P under conjugation and then using the algorithm to test whether $[Q] \subseteq [P]$.

The algorithm maintains two sets $R, T \subseteq S \times S \times S$. The former keeps record of the elements that are added to T during the course of the algorithm. To simplify the presentation, we define $x \cdot a^{-1}$ to be the set of all elements $p \in S^1$ which satisfy the equation $p \cdot h(a) = x$.

Algorithm 2 Testing for strong recognition

```
initialize R and T with the set {(s, e, 1) | (s, e) ∈ P}
while T ≠ ∅ do
    remove some element (s, x, y) from T
    if x = 1 then return "[P] ⊈ [Q]" end if
    if (sx, yxyx) ∉ Q then
        for all a ∈ A, p ∈ x · a^{-1} do
            if (s, p, h(a)y) ∉ R then add (s, p, h(a)y) to R and to T end if
        end for
    end if
end while
return "[P] ⊆ [Q]"
```

The following technical Lemma is crucial for the correctness proof of the algorithm:

▶ **Lemma 5.** *Let $u, v \in A^+$ and let (s, e) and $(h(u), h(v))$ be linked pairs. Then uv^ω is contained in $[s][e]^\omega$ if and only if there exists a factorization $v = v_1v_2$ such that $v_1 \neq \varepsilon$, $h(uv_1) = s$ and $h(v_2vv_1) = e$.*

Proof. Let $v = a_1a_2\cdots a_n$ with $n \geqslant 1$ and $a_i \in A$. If uv^ω is contained in $[s][e]^\omega$, there exists a factorization $uv^\omega = u'v_1'v_2'\cdots$ such that $h(u') = s$ and $h(v_i') = e$ for all $i \geqslant 1$. Since u and v are finite words, there exist indices $j > i \geqslant 1$, powers $k, \ell \geqslant 1$ and a position $m \in \{1, \ldots, n\}$ such that $u'v_1'v_2'\cdots v_{i-1}' = uv^k a_1a_2\cdots a_m$ and $v_i'v_{i+1}'\cdots v_j' = a_{m+1}a_{m+2}\cdots a_n v^\ell a_1a_2\cdots a_m$. We set $v_1 = a_1a_2\cdots a_m$ and $v_2 = a_{m+1}a_{m+2}\cdots a_n$. Then $v_1v_2 = v$,

$$\begin{aligned} h(uv_1) &= h(uv^k a_1a_2\cdots a_m) &&= h(u'v_1'v_2'\cdots v_{i-1}') = se^{i-1} &&= s \text{ and} \\ h(v_2vv_1) &= h(a_{m+1}a_{m+2}\cdots a_n v^\ell a_1a_2\cdots a_m) &&= h(v_i'v_{i+1}'\cdots v_j') &&= e^{j-i+1} = e. \end{aligned}$$

To prove the converse direction, consider the factorization $uv^\omega = uv_1(v_2vv_1)^\omega$. ◀

To simplify the proofs of the following two Lemmas, we extend h to a monoid morphism $h^1\colon A^* \to S^1$ by setting $h^1(u) = h(u)$ for all $u \in A^+$ and $h^1(\varepsilon) = 1$.

▶ **Lemma 6.** *If the difference $[P] \setminus [Q]$ is non-empty, the algorithm returns "$[P] \not\subseteq [Q]$".*

Proof. By the closure properties of regular languages, we know that there exists a word $\alpha = u(a_1a_2\cdots a_n)^\omega \in [P] \setminus [Q]$. Let $s = h(u)$ and $e = h(a_1a_2\cdots a_n)$. Lemma 5 shows that we can assume without loss of generality that (s, e) is contained in P. We now prove by induction on

the parameter k that upon termination, we have $(s, h^1(a_1a_2\cdots a_k), h^1(a_{k+1}a_{k+2}\cdots a_n)) \in R$ for all $k \in \{0,\ldots,n\}$. In particular, by considering the case $k = 0$, we see that the element $(s, 1, e)$ is added to R. Since every element added to R is also added to Q, the algorithm returns "$[P] \not\subseteq [Q]$".

The base case $k = n$ is covered by the initialization of the set R. Let now $k < n$, $x = h^1(a_1a_2\cdots a_{k+1})$ and $y = h^1(a_{k+2}a_{k+3}\cdots a_n)$. By the induction hypothesis, we know that the tuple (s, x, y) is added to T during the course of the algorithm. Consider the iteration when this tuple is removed from T. Because of $\alpha \notin [Q]$, we know that $(sx, yxyx) \notin Q$. Thus the inner loop guarantees that $(s, h^1(a_1a_2\cdots a_k), h^1(a_{k+1}a_{k+2}\cdots a_n))$ is added to R. ◀

▶ **Lemma 7.** *If the algorithm returns "$[P] \not\subseteq [Q]$", the difference $[P] \setminus [Q]$ is non-empty.*

Proof. We construct a word in the difference $[P] \setminus [Q]$. For every triple $(s, e, 1)$ that is added to R during the initialization, we define $w[s, e, 1] = \varepsilon$. If a triple $(s, p, h(a)y)$ is added to R later, we set $w[s, p, h(a)y] = a \cdot w[s, p \cdot h(a), y]$. For every $(s, x, y) \notin R$, the word $w[s, x, y]$ is undefined. If $w[s, x, y]$ is defined, its image under h^1 is y and we have $(s, xy) \in P$. Both properties are easy to prove by induction.

Let $(s, 1, y)$ be the triple that was removed from T immediately before the termination of the algorithm. Consider an arbitrary word $u \in [s]$ and set $v = w[s, 1, y]$. We have $(s, y) \in P$ and thus $uv^\omega \in [P]$. For every factorization $v = v_1av_2$ where $v_1, v_2 \in A^*$ and $a \in A$, the word $w[s, h^1(v_1), h^1(av_2)]$ is defined as av_2 and thus, the tuple $(h(uv_1a), h(v_2vv_1a))$ is not contained in Q. In view of Lemma 5, this shows that $uv^\omega \notin [Q]$. ◀

We are now able to state the main result of this section:

▶ **Theorem 8.** *Given a morphism $h\colon A^+ \to S$ onto a finite semigroup S and two sets of linked pairs $P, Q \subseteq S \times E(S)$, one can check in $\mathcal{O}(|A| \cdot |S|^3)$ time whether $[P] \subseteq [Q]$.*

Proof. The correctness of Algorithm 2 follows from the previous two Lemmas. Since R contains at most $(|S|+1)^3$ elements when the algorithm terminates, the outer loop is executed at most $(|S|+1)^3$ times. Moreover, for all $a \in A$ and $s, t \in S$ with $s \neq t$, the sets $s \cdot a^{-1}$ and $t \cdot a^{-1}$ are disjoint. Thus, each element $p \in S^1$ is considered at most $|A| \cdot (|S|+1)^2$ times in the inner loop. If R is implemented as a bit field and T is implemented as a linked list, all operations take constant time. This shows that the total running time is in $\mathcal{O}(|A| \cdot |S|^3)$. ◀

6 Computation of the syntactic morphism

In this section, we present an algorithm to compute the syntactic semigroup for a given language. The syntactic homomorphism is obtained as a byproduct. One can show that the syntactic semigroup is the smallest semigroup strongly recognizing a language [1, 9], so this operation is in some sense analogous to minimization of finite automata. The most important difference is that our algorithm requires only quadratic time, whereas minimization is PSPACE-hard in the case of Büchi automata [8, 12].

Let S be a finite semigroup, let $h\colon A^+ \to S$ be a surjective morphism and let P be a set of linked pairs that is closed under conjugation. To make the following notation more readable, we define Q to be the maximal subset of $S \times S$ such that $[P] = [Q]$.

▶ **Lemma 9.** *Let $u, v \in A^+$. Then $uv^\omega \in [P]$ if and only if $(h(u), h(v)) \in Q$.*

Proof. Suppose that $uv^\omega \in [P]$. By Proposition 1 we have $[h(u)][h(v)]^\omega \subseteq [P] = [Q]$. Since Q is maximal, the pair $(h(u), h(v))$ is contained in Q. The converse implication is trivial. ◀

We now define a equivalence relation $\cong$ on S by $s \cong t$ if for all $z \in S$, we have

$$(z,s) \in Q \Leftrightarrow (z,t) \in Q \text{ and}$$
$$(s,z) \in Q \Leftrightarrow (t,z) \in Q.$$

Moreover, let $\equiv$ be the coarsest congruence on S that refines $\cong$, i.e., $s \equiv t$ if $xsy \cong xty$ for all $x, y \in S^1$. We denote by $[s]_\equiv$ the equivalence class $\{t \in S \mid t \equiv s\}$ of an element $s \in S$. The relation $\equiv$ is closely related to the syntactic congruence, as confirmed by the following result:

▶ **Proposition 10.** *The quotient semigroup $S/\equiv$ is isomorphic to $A^+/\equiv_L$.*

Proof. We first define a morphism $g\colon A^+ \to S/\equiv$ by setting $g(u) = [h(u)]_\equiv$ for all $u \in A^+$. Let now $u, v \in A^+$. By Lemma 9, we have $h(u) \equiv h(v)$ if and only if $h_L(u) = h_L(v)$. Thus, $g \circ h_L^{-1}$ is a semigroup isomorphism. ◀

The computation of the syntactic semigroup requires two steps:

1. Compute the partition induced by the equivalence relation $\cong$.
2. Refine the partition until the underlying equivalence relation becomes a congruence.

The first step can be performed in time quadratic in the size of the semigroup. For the second step, we can adapt Hopcroft's minimization algorithm for finite automata [6]. For $C \subseteq S$ and $a \in A$, we define

$$C \cdot a^{-1} = \{s \in S \mid s \cdot h(a) \in C\} \quad \text{and} \quad a^{-1} \cdot C = \{s \in S \mid h(a) \cdot s \in C\}.$$

The full algorithm is shown in Algorithm 3. It relies on the Split routine that is usually implemented as part of a *partition refinement data structure*, see e.g. [6] for details. Its semantics is shown in Algorithm 4. In addition to modifying the partition, that routine also updates a set $T \subseteq 2^S$ that is used in the main algorithm.

Algorithm 3 Computing the syntactic semigroup

```
initialize a partition with a single class S
for all s ∈ S do
    Split({t ∈ S | (s,t) ∈ Q})
    Split({t ∈ S | (t,s) ∈ Q})
end for
initialize T with the non-trivial classes of the partition
while T ≠ ∅ do
    remove some set C from T
    for all a ∈ A do
        Split(C · a^{-1})          ▷ Refine the partition and update T
        Split(a^{-1} · C)          ▷ Refine the partition and update T
    end for
end while
```

The next Lemma shows that upon termination, the equivalence relation induced by the partition is indeed a congruence:

▶ **Lemma 11.** *If, upon termination, the elements s and t belong to the same class of the partition, then, for each $a \in A$, the elements $h(a)s$ and $h(a)t$ are in the same class as well.*

Algorithm 4 The Split operation to refine a partition $\mathcal{P}$

```
procedure Split(X)
    for all C ∈ P do
        C1 ← C ∩ X, C2 ← C \ X
        if C1 ≠ ∅ and C2 ≠ ∅ then
            P ← (P \ {C}) ∪ {C1, C2}
            if C ∈ T then
                T ← (T \ {C}) ∪ {C1, C2}
            else
                if |C1| ⩽ |C2| then T ← T ∪ {C1} else T ← T ∪ {C2} end if
            end if
        end if
    end for
end procedure
```

Proof. Suppose that $h(a) \cdot s$ and $h(a) \cdot t$ belong to different classes. These elements are split either during the initialization or in the main loop. In either case, a set C that contains either $h(a) \cdot s$ or $h(a) \cdot t$ is added to T. When this set is removed from T, the operation $\mathrm{Split}(a^{-1} \cdot C)$ asserts that s and t lie in different classes as well. A dual argument holds in the right-sided case. ◀

There is of course a dual statement for the elements $s \cdot h(a)$ and $t \cdot h(a)$.

▶ **Theorem 12.** *The syntactic morphism can be computed in $\mathcal{O}(|S|^2 + |A| \cdot |S| \log |S|)$ time.*

Proof. Let us first argue that Algorithm 3 is correct. The partition is initialized with the equivalence classes of $\cong$. A class is only split when it is necessary to restore the left-stability or right-stability. Upon termination, the relation induced by the partition is a congruence, as stated in Lemma 11. Thus, it is the coarsest congruence that refines $\cong$ and hence equivalent to $\equiv$.

For the analysis of the running time, we assume that the operation $\mathrm{Split}(X)$ can be implemented in time linear in $|X|$. Then the initialization clearly takes $\mathcal{O}(|S|^2)$ time. We denote by $C_1, \ldots, C_k$ the sets that are added to T during the course of the algorithm. Let $s \in S$ and let $n_s = \{i \mid 1 \leqslant i \leqslant k, s \in C_i\}$ be the number of sets C_i containing s. At any point in time, there is at most one set in T that contains s. If such a set C is removed from T and another set C' with $s \in C'$ is added to T at a later point in time, we have that $|C'| \leqslant |C|/2$. Thus, the inequality $n_s \leqslant \log |S|$ holds for all $s \in S$ and we have

$$\sum_{i=1}^{k} \sum_{a \in A} \left(\left| C_i \cdot a^{-1} \right| + \left| a^{-1} \cdot C_i \right| \right) = \sum_{s \in S, a \in A} \left(n_{s \cdot h(a)} + n_{h(a) \cdot s} \right) \leqslant 2\,|A| \cdot |S| \log |S|.$$

Consequently, the total running time of the *while*-loop is in $\mathcal{O}(|A| \cdot |S| \log |S|)$, assuming that T is implemented efficiently, e.g. as a linked list. ◀

If the alphabet A is fixed and the semigroup S becomes large, the running time is dominated by the initialization. However, the following result implies that the algorithm we presented is quite optimal.

▶ **Proposition 13.** *Let $k \in \mathbb{N}$ and let $\lambda \in \mathbb{R}$ be a strictly positive number. One cannot compute the syntactic morphism in time $\mathcal{O}(|A|^k \cdot |S|^{2-\lambda})$.*

Proof. We assume that there exists a deterministic algorithm and a constant $c \geqslant 1$, such that every input of size $n = |S|$ and $m = |A|/2$ can be minimized in time $T(n,m) \leqslant c \cdot m^k \cdot n^{2-\lambda}$. Since c, k and λ are constant, there exists an integer $m \in \mathbb{N}$ with $2^{\lambda m} > 16c \cdot m^k$. Consider an alphabet $A = \{1, \dots, 2m\}$ satisfying this condition.

We define $A_1 = \{1, \dots, m\}$ and $A_2 = \{m+1, \dots, 2m\}$, $S_1 = (2^{A_1} \setminus \{\emptyset\}) \times \{\emptyset\}$ and $S_2 = \{\emptyset\} \times (2^{A_2} \setminus \{\emptyset\})$. The set $S = S_1 \cup S_2$ forms a semigroup with the multiplication

$$(X_1, X_2) \cdot (Y_1, Y_2) = \begin{cases} (\emptyset, X_2 \cup Y_2) & \text{if } X_1 = Y_1 = \emptyset \\ (X_1 \cup Y_1, \emptyset) & \text{otherwise} \end{cases}$$

Furthermore, let $h\colon A^+ \to S$ be defined by $h(a) = (\{a\}, \emptyset)$ for all $a \in A_1$ and $h(b) = (\emptyset, \{b\})$ for all $b \in A_2$. Let F be the set of linked pairs of S. It is easy to verify that $S_1 \times S_2 \subseteq F$. Moreover, two tuples $(s,e), (t,f) \in S_1 \times S_2$ are conjugate if and only if $(s,e) = (t,f)$. The number of conjugacy classes of S is at least $|S_1| \cdot |S_2| \geqslant 2^{m-1} \cdot 2^{m-1} = 4^{m-1}$. The size of S is $n = |S_1| + |S_2| \leqslant 2^m + 2^m = 2^{m+1}$.

Consider the execution of the algorithm on input h and $P = F$. Since $[P] = A^\omega$, the algorithm returns the trivial semigroup. We denote by $(s_1, e_1), (s_2, e_2), \dots, (s_\ell, e_\ell)$ the sequence of linked pairs for which the algorithm checks whether $(s_i, e_i) \in P$. We have $\ell \leqslant T(n,m) \leqslant c \cdot m^k \cdot n^{2-\lambda} < 4c \cdot m^k \cdot 2^{2m-\lambda m} = 16c \cdot m^k \cdot 2^{-\lambda m} \cdot 4^{m-1} < 4^{m-1}$ and thus, there is a conjugacy class C such that $(s_i, e_i) \notin C$ for all $i \in \{1, \dots, \ell\}$. Since the algorithm is deterministic, the execution sequence on input $Q = P \setminus C$ is the same, and the algorithm returns, again, the trivial semigroup consisting of one element. However, $[Q] \neq A^\omega$ and thus, the algorithm is incorrect. ◀

One can also show that, independent of the alphabet size, it is impossible to compute the syntactic morphism in time $\mathcal{O}(|S|^{2-\lambda})$. However, the proof is a bit more involved [4].

7 Language operations on morphisms

One of the merits of strong recognition is that complementation is easy. If a morphism $h\colon A^+ \to S$ onto a finite semigroup S strongly recognizes a language $L \subseteq A^\omega$, it also strongly recognizes the complement $A^\omega \setminus L$. As in the case of finite words, we can use *direct products* for unions and intersections.

Another operation on languages which is of particular interest when it comes to converting MSO formulas to strongly recognizing morphisms are so-called *length-preserving morphisms*. Suppose we are given alphabets A, B and a length-preserving morphism $\pi\colon A^+ \to B^+$, i.e., $\pi(a) \in B$ for all $a \in A$. We naturally extend this morphism to infinite words by setting $\pi(a_1 a_2 \cdots) = \pi(a_1)\pi(a_2)\cdots$ and to languages $L \subseteq A^\omega$ by setting $\pi(L) = \{\pi(\alpha) \mid \alpha \in L\}$.

▶ **Proposition 14.** *Let $\pi\colon A^+ \to B^+$ be a length-preserving morphism, let S be a finite semigroup and let $h\colon A^+ \to S$ be a surjective morphism that strongly recognizes a language $L \subseteq A^\omega$. Then there exist a semigroup T of size $2^{|S|}$ and a morphism $g\colon B^+ \to T$ that strongly recognizes $\pi(L)$.*

Proof. We first define T to be the set 2^S of all subsets of S and extend it to a semigroup by defining an associative multiplication $X \cdot Y = \{xy \mid x \in X, y \in Y\}$. The morphism $g\colon B^+ \to T$ is uniquely defined by $g(a) = h(\pi^{-1}(a))$ for all $a \in B$.

Let us now verify that g strongly recognizes $\pi(L)$. Consider a linked pair (s,e) and two infinite words $\alpha, \beta \in g^{-1}(s)(g^{-1}(e))^\omega$. By Proposition 1, it suffices to show that $\alpha \in \pi(L)$ implies $\beta \in \pi(L)$. If α is contained in $\pi(L)$, we can conclude by Ramsey's theorem that

Table 1 Experimental results for different parameter values.

	φ_k			ψ_k			χ_k		
	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert P\rvert$	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert P\rvert$	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert P\rvert$
$k=2$	4	5	1	12	15	10	7	14	11
$k=3$	8	22	1	43	50	41	11	26	15
$k=4$	16	74	1	148	163	146	17	61	30
$k=5$	32	232	1	539	570	537	41	227	85
$k=6$	64	710	1	1863	1926	1861	105	716	184

there exists a linked pair (t, f) of S with $t \in s$, $f \in e$ and $h^{-1}(t)(h^{-1}(f))^\omega \cap L \neq \emptyset$. By assumption, h strongly recognizes L and thus, we have $h^{-1}(t)(h^{-1}(f))^\omega \subseteq L$. Since we know that there exists an infinite word $uv_1v_2\cdots \in \pi^{-1}(\beta)$ such that $h(u) = t$ and $h(v_i) = f$ for all $i \geqslant 1$, this immediately yields $uv_1v_2\cdots \in L$ and hence $\beta \in \pi(L)$. ◀

8 Experimental results

In order to test the algorithms and constructions in practice, we implemented the conversion of MSO formulas into strongly recognizing morphisms. The constructions described in Section 7 are used to recursively convert the formulas, and all intermediate results are minimized using the algorithm from Section 6. For details on MSO logic over infinite words and its connexion to regular languages, we refer to [15, 16]. The conversion to strongly recognizing morphisms instead of Büchi automata has the advantage that all intermediate objects can be minimized efficiently. Table 1 shows the size of the computed syntactic semigroup S, the number of linked pairs F and the size of the accepting set P (which is closed unter conjugation) for the following three families of MSO formulas with parameter $k \geqslant 1$ and free second-order variables $X_{k+1} = X_1, X_2, \ldots, X_k$:

$$\varphi_k = \forall x \bigwedge_{i=1}^{k} \exists y\, (x < y \wedge y \in X_i)$$

$$\psi_k = \forall x \forall y\, (y = x+1) \to \bigwedge_{i=1}^{k} (x \in X_i \to y \in X_{i+1})$$

$$\chi_k = \forall x \bigwedge_{i=1}^{k} (x \in X_i \to \exists y\, (x < y \wedge (y \in X_{i-1} \vee y \in X_{i+1})))$$

All computations were made on a Intel Core i5-3320M with 4GiB of RAM. The execution time was less than three seconds for each formula.

9 Summary and Outlook

We described several algorithms for weakly recognizing morphisms and strongly recognizing morphisms over infinite words. Our tests indicate that strongly recognizing morphisms, when combined with the minimization algorithm presented in Section 6, are a practical alternative to automata-based models when it comes to deciding properties of MSO formulas.

Some of the algorithms leave room for optimization. In particular, it would be interesting to see whether there is a linear-time algorithm to compute conjugacy classes and whether the running time of the algorithm described in Section 5 can be improved to $\mathcal{O}(|A| \cdot |S^2|)$.

References

1 A. Arnold. A syntactic congruence for rational ω-languages. *Theoretical Comput. Sci.*, 39:333–335, 1985.

2 H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *MFCS 94, Proceedings*, volume 802 of *LNCS*, pages 554–566. Springer, 1994.

3 V. Diekert and P. Gastin. First-order definable languages. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 261–306. Amsterdam University Press, 2008.

4 L. Fleischer and M. Kufleitner. Efficient Algorithms for Morphisms over Omega-Regular Languages. *CoRR*, abs/1509.06215, 2015.

5 V. Froidure and J.-É. Pin. Algorithms for computing finite semigroups. In F. Cucker and M. Shub, editors, *Foundations of Computational Mathematics*, pages 112–126. Springer, 1997.

6 J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, New York, 1971.

7 J. Hopcroft and R. Karp. A linear algorithm for testing equivalence of finite automata. Technical report, Dept. of Computer Science, Cornell Univ., December 1971.

8 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.

9 D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

10 J.-P. Pécuchet. Varietés de semisgroupes et mots infinis. In *STACS 86*, volume 210 of *LNCS*, pages 180–191. Springer, 1986.

11 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959. Reprinted in E. F. Moore, editor, *Sequential Machines: Selected Papers*, Addison-Wesley, 1964.

12 A. P. Sistla, M. Y. Vardi, and P. L. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Comput. Sci.*, 49(2-3):217–237, 1987.

13 L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, TR 133, M.I.T., Cambridge, 1974.

14 R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, Apr. 1975.

15 W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier, 1990.

16 W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 389–455. Springer, Berlin, 1997.

Approximating the Regular Graphic TSP in Near Linear Time

Ashish Chiplunkar[1,2] and Sundar Vishwanathan[2]

1 **Amazon Development Center**
Bangalore, India
ashish.chiplunkar@gmail.com

2 **Department of Computer Science and Engineering**
Indian Institute of Technology Bombay
Mumbai, India
sundar@cse.iitb.ac.in

Abstract

We present a randomized approximation algorithm for computing traveling salesperson tours in undirected regular graphs. Given an n-vertex, k-regular graph, the algorithm computes a tour of length at most $\left(1+\frac{4+\ln 4+\varepsilon}{\ln k-O(1)}\right)n$, with high probability, in $O(nk\log k)$ time. This improves upon the result by Vishnoi ([27], FOCS 2012) for the same problem, in terms of both approximation factor, and running time. Furthermore, our result is incomparable with the recent result by Feige, Ravi, and Singh ([10], IPCO 2014), since our algorithm runs in linear time, for any fixed k. The key ingredient of our algorithm is a technique that uses edge-coloring algorithms to sample a cycle cover with $O(n/\log k)$ cycles, with high probability, in near linear time.

Additionally, we also give a deterministic $\frac{3}{2}+O\left(\frac{1}{\sqrt{k}}\right)$ factor approximation algorithm for the TSP on n-vertex, k-regular graphs running in time $O(nk)$.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases traveling salesperson problem, approximation, linear time

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.125

1 Introduction

Given a complete undirected graph with positive real valued weights on the edges, the traveling salesperson problem (TSP) is to find a minimum weight cycle that visits each vertex exactly once. This problem was among the first few proved NP-Complete by Karp [15]. In the absence of any structural restriction on the weight function, the TSP is hard to approximate within any constant factor ([26], [24]).

The most widely researched restriction of the TSP is the MetricTSP, where the vertices form a metric space with the weight function as the metric. This simple imposition of the triangle inequality over the weights allowed Christofides [7] to efficiently construct tours with an approximation ratio of 3/2. No improvement has been made on this upper bound in the last 35 years. However, for the case when the metric is Euclidean with a fixed number of dimensions (the EuclideanTSP), polynomial time approximation schemes are known [1, 18, 23, 3].

The possibility of existence of a polynomial time approximation scheme for the MetricTSP was ruled out early on by the proof of its APX-hardness given by Papadimitriou and Yannakakis [22]. The first explicitly proven lower bound on the approximation factor was 5381/5380 by Engebretsen [9] (for the MetricTSP with distances 1, and 2). This was

followed by a series of improvements: 3813/3812 by Böckenhauer and Seibert [5], 220/219 by Papadimitriou and Vempala [21], 185/184 by Lampis [17], and finally, 123/122 by Karpinski, Lampis, and Schmied [16], which is the best lower bound known currently. The reader is referred to [16] for a nice overview of recent advances in many natural restrictions of the MetricTSP.

An important sub-class of the MetricTSP is the GraphTSP, where the weight function on the edges arises from the shortest path distances in some unweighted undirected graph. This is believed to be the most promising candidate for capturing the computational hardness of the MetricTSP. GraphTSP is APX-hard too, as a consequence of its MAX-SNP hardness [22] and the PCP theorem [2]. The best known lower bound of 4/3 on the integrality gap of the Held-Karp LP relaxation [13] of the MetricTSP is observed on an instance of the GraphTSP.

Gharan, Saberi and Singh [12] achieved the first improvement over Christofides [7] algorithm for the GraphTSP with an approximation ratio strictly less than 3/2, which was shortly followed by Mömke and Svensson's [19] bound of 1.461. Mucha [20] later improved the analysis of Mömke and Svensson's [19] algorithm and demonstrated a bound of 13/9. Currently, the best known bound is 7/5, given by Sebö and Vygen [25]. It is widely believed that the Held-Karp relaxation has an integrality gap of precisely 4/3, and this has been proven for cubic graphs [6].

Vishnoi [27] opened up a new line of interesting work by arguing that approximating the GraphTSP might possibly get better with increasing edge density. He studied the GraphTSP on regular graphs (the RegGraphTSP), and proved that approximation factors arbitrarily close to 1 can be achieved, as the degree of the regular graph becomes larger. The reader is referred to to Vishnoi [27] for a nice survey on the MetricTSP in general, and an interesting discussion on this line of work.

The main technical contribution of Vishnoi's paper is an algorithm for the RegGraphTSP with an approximation factor of $(1 + \sqrt{64/\ln k})$ on regular graphs with degree k. Given a k-regular graph with n vertices, the algorithm first samples a cycle cover using Jerrum, Sinclair and Vigoda's algorithm [14] for sampling a matching from an almost uniform distribution over the perfect matchings in the natural bipartite version of the input graph. This cycle cover is guaranteed to have $O(n/\sqrt{\ln k})$ cycles with high probability. These cycles are then connected using two copies of a spanning tree on the graph formed by contracting the cycles. This yields a tour of length at most $(1 + \sqrt{64/\ln k})n$ with probability $1 - 1/n$. The running time of this algorithm is dictated by the running time of the sampling method, which is around $O(n^{10} \log^3 n)$. This can be improved marginally by using a faster sampling algorithm, for example, the algorithm by Bezáková, Stefankovic, Vazirani and Vigoda [4].

In a follow-up paper, Feige, Ravi, and Singh [10] improve the approximation ratio for the RegGraphTSP to $1 + O(1/\sqrt{k})$. They use a randomized procedure to construct vertex disjoint paths in the input graph which, in expectation, contain $(1 - O(1/\sqrt{k}))n$ edges. They connect these paths arbitrarily using another $O(n/\sqrt{k})$ edges, resulting in a tree with $O(n/\sqrt{k})$ vertices of odd degree. Then they show that these vertices can be matched with paths of total length $O(n/\sqrt{k})$, that is, they have a *T-join* of size $O(n/\sqrt{k})$, resulting in an Eulerian graph. Short-cutting an Euler tour of this graph yields a $(1 + O(1/\sqrt{k}))$-approximation. The running time of this algorithm is dictated by the time taken to find the T-join, which is $O(n^3)$.

Here we propose an alternative method for solving the RegGraphTSP, which achieves an approximation factor better than Vishnoi's. More importantly, our algorithm runs in linear time, for every fixed k.

▶ **Theorem 1.** *Fix an $\varepsilon > 0$. There is an algorithm which, given a connected k-regular undirected graph on n vertices, runs in time $O(nk \log k)$, and outputs a TSP tour of cost at most $\left(1 + \frac{4+\ln 4+\varepsilon}{\ln(k/2)}\right) n$ with high probability (specifically, probability of failure decaying exponentially with n).*

The idea behind improving the running time is to replace the Jerrum-Sinclair-Vigoda subroutine in Vishnoi's algorithm by a much faster sampling subroutine. Although the Jerrum-Sinclair-Vigoda algorithm comes with stringent guarantees about the resulting sampling distribution, such guarantees are not requisite for Vishnoi's algorithm. On the other hand, while our sampling distribution on the cycle covers may be quite far from uniform, we demonstrate bounds on the measure concentration around cycle covers with few cycles, using simple counting arguments. We describe the algorithm in Section 2, and analyze it in Section 3.

While derandomizing our algorithm seems like a difficult problem, we also have a simple deterministic linear time algorithm that achieves a $\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)$ factor approximation. Here, the main idea is to traverse the graph in a depth-first-like manner and keep removing long cycles. These cycles cover a good fraction of the vertices. The cycles and the uncovered vertices can then be connected by a spanning tree. We devote Section 4 for this algorithm and its analysis.

2 The Randomized Algorithm

The high level idea behind our algorithm is similar to that of Vishnoi's. Find a cycle cover of the graph, and then connect the cycles using a spanning tree. Recall that a cycle cover of a graph is a collection of vertex-disjoint cycles that cover all its vertices. We wish to construct a cycle cover such that it has a small number of cycles with high probability. It is folklore that cycle covers in a graph correspond to matchings in the natural encoding of the given graph as a bipartite graph (see Definition 3). Indeed, Vishnoi selects a random matching in such an encoding.

Given a k-regular graph, we intend to first partition the edges into cycle covers in a randomized manner, and then select the best cycle cover. Our algorithm to find the partition uses ideas from the Gabow-Kariv algorithm [11], which finds a minimum edge-coloring of an input graph. However, the Gabow-Kariv algorithm works only on graphs with vertex degrees which are powers of two. Therefore, we attempt to reduce the degree to a power of two, for which we need to work with directed regular graphs and their bipartite encodings.

▶ **Definition 2.** We say that a directed graph is *k-regular* if the in-degree as well as the out-degree of each vertex is k. A *cycle cover* in a directed graph is a 1-regular subgraph of the graph.

▶ **Definition 3.** The *bipartite encoding* of a directed graph $G = (V, A)$ is the bipartite graph $B = (V_L, V_R, E)$, where V_L and V_R contain vertices v_L and v_R respectively, for each $v \in V$, and E contains the edge $\{u_L, v_R\}$ for each arc $(u, v) \in A$.

From the definition, it is easy to see a natural bijection between the cycle covers of a directed graph and perfect matchings of its bipartite encoding. Analogously, our algorithm to partition the arcs of a regular directed graph into cycle covers can also be seen as an algorithm to partition the edges of a regular bipartite graph into perfect matchings.

The reason for working with directed graphs is that one can effectively partition the edges of a k-regular directed graph into k cycle covers. As a consequence, we have the following

lemma which ensures there is no loss of generality if we restrict our attention to the case where the degree k is a power of two. This lemma relies on the algorithm by Cole, Ost, and Schirra [8], which partitions the edges of any given k-regular bipartite undirected graph with n vertices into perfect matchings, and runs in time $O(nk \log k)$.

▶ **Lemma 4.** *Given a K-regular directed graph $G' = (V, A')$ with n vertices and $k < K$, there is an algorithm which outputs a k-regular subgraph $G = (V, A)$ of G', and runs in time $O(nK \log K)$.*

Proof. The algorithm constructs the bipartite encoding B' of G'. Therefore, B' is a K-regular bipartite graph. The algorithm then partitions the edges of B' into K perfect matchings, using the Cole-Ost-Schirra algorithm, and then deletes an arbitrary set of $K - k$ matchings. This gives a k-regular bipartite subgraph B of B'. The algorithm returns $G = (V, A)$, where $A \subseteq A'$ consisting of arcs which correspond to edges in B. ◀

Henceforth, we will assume that k is a power of 2. Otherwise, if $2^l < k < 2^{l+1}$ for some $l \in \mathbb{N}$, we preprocess the given graph using the algorithm from Lemma 4 to obtain a 2^l-regular subgraph. We randomly partition the arcs of the subgraph into cycle covers, and then pick the best cycle cover.

▶ **Definition 5.** Let $G = (V, A)$ be a k-regular directed graph. A *cycle cover coloring* of this graph is an ordered partition of the arc set A into k cycle covers. Formally, it is a function $c : A \longrightarrow \{1, \ldots, k\}$, such that for each $i \in \{1, \ldots, k\}$, the set $c^{-1}(i)$ is a cycle cover of G.

In other words, for any vertex v and color i, exactly one arc leaving v and exactly one arc entering v have color i. In fact, the algorithm from Lemma 4 creates an arbitrary such coloring. Thus, regular directed graphs have efficiently constructible cycle cover colorings. However, the cycle cover coloring resulting from the degree reduction algorithm might not contain a cycle cover with a small number of cycles. To address this issue, we next describe a procedure to construct a random cycle cover coloring of a k-regular graph. This falls into the "divide and conquer" paradigm, where the "conquer" step involves partitioning the edges of a 2-regular directed graph into two cycle covers, and relies on the following lemma.

▶ **Lemma 6.** *The arcs of a 2-regular directed graph can be partitioned into two cycle covers in linear time.*

Proof. Construct the bipartite encoding of the 2-regular graph. Since the in-degree and out-degree of each vertex in the bipartite graph is two, the bipartite encoding is a 2-regular undirected graph, that is, a collection of vertex disjoint cycles of even length. Partition the edges of the bipartite encoding into two perfect matchings. Each of these two matchings encodes a cycle cover of the original directed graph. ◀

Our procedure to generate a random cycle cover coloring of a given k-regular directed graph, which forms the heart of the approximation algorithm claimed in Theorem 1, is given by Algorithm 1, and we call it RandCycleCoverColoring. It is easily verified that the running time $T(n, k)$ of RandCycleCoverColoring on a k-regular graph with n vertices is given by the recurrence $T(n, k) = T(2n, k/2) + O(nk)$. This yields $T(n, k) = O(nk \log k)$. Theorem 7 states that the random cycle cover coloring contains, with high probability, a cycle cover with a small number of components. The proof is deferred to the next section.

▶ **Theorem 7.** *Fix an $\varepsilon > 0$. Let G be a k-regular directed graph with n vertices, where k is a power of 2. The algorithm* RandCycleCoverColoring*, on input G, outputs a random cycle cover coloring of G, which with high probability contains a cycle cover with at most $(2 + \ln 2 + \varepsilon)n / \ln k$ components. The algorithm runs in time $O(nk \log k)$.*

Algorithm 1 RandCycleCoverColoring(G)

1: {INPUT: G, a k-regular n vertex directed graph with k being a power of 2; OUTPUT: A random cycle cover coloring of G.}
2: If $k = 1$ **return** G with each arc colored 1.
3: Convert G into a $k/2$-regular digraph $H = (V', A')$ with $2n$ vertices, by splitting every vertex v into a pair of vertices: v_0 and v_1. Distribute the arcs incident on v randomly among v_0 and v_1, so that each gets half of the incoming and half of the outgoing arcs.
4: Recursively call RandCycleCoverColoring(H) to obtain an edge coloring $c' : A' \longrightarrow \{1, \ldots, k/2\}$ of H.
5: Fuse the pairs of vertices back to obtain G with the coloring c'. For each i, the edges colored i constitute a 2-regular directed graph. Call it G_i.
6: For each $i \in \{1, \ldots, k/2\}$, partition the arcs of G_i into two cycle covers, using Lemma 6. Recolor one of these cycle covers with color $i + k/2$.

It is worth noting that failure probability of RandCycleCoverColoring decays exponentially with n, and the parameter ε only affects the rate of this decay. The algorithm itself (and hence, its running time) is independent of ε.

Theorem 1 follows from Theorem 7 in the following manner. Given a connected K-regular undirected graph over the vertex set V of size n, construct the directed graph $G' = (V, A')$ in the obvious manner: for each edge $\{u, v\}$ of the undirected graph, include the arcs (u, v) and (v, u) in A'. Clearly, G' is a K-regular directed graph. Use the degree reduction algorithm from Lemma 4 to get a regular graph $G = (V, A)$ with degree $k = 2^{\lfloor \log_2 K \rfloor}$. Now run the procedure RandCycleCoverColoring on G to get a random cycle cover coloring of G. Choose the best cycle cover from this cycle cover coloring. This cycle cover contains at most $(2 + \ln 2 + \varepsilon)n / \ln k$ cycles, with high probability.

The rest of the processing is routine. Take the multi-set E of edges in the original graph which correspond to the arcs constituting the cycle cover. (If both arcs (u, v) and (v, u) belong to the cycle cover, then take edge $\{u, v\}$ with multiplicity two.) Contract these edges, and find a spanning tree of the resulting minor. Duplicate the edges of the spanning tree, so that these edges and the edges in E form an Eulerian spanning subgraph of G. Find an Euler tour in this graph and short-cut it to get a TSP tour of G. The cost of this tour is at most $n + 2 \times \frac{(2+\ln 2+\varepsilon)n}{\ln k} \leq \left(1 + \frac{4+\ln 4+2\varepsilon}{\ln(K/2)}\right) n$, and this post-processing can be done in time $O(nk)$, that is, linear in the size of the graph.

3 Analysis of RandCycleCoverColoring

We first bound from above the probability of getting any fixed cycle cover coloring.

▶ **Lemma 8.** *Consider a fixed cycle cover coloring c of the k-regular directed graph $G' = (V, A)$, where k is a power of 2, let and $n = |V|$. The probability that* RandCycleCoverColoring*, on input G', outputs c is at most $f(n, k)$, where*

$$f(n, k) = \left[\frac{k^k}{(k!)^2}\right]^n$$

Proof. By induction on k. The claim is trivial for $k = 1$. Assume now that $k > 1$. Consider the coloring $c' : A \longrightarrow \{1, \ldots, k/2\}$ given by

$$c'(e) = \begin{cases} c(e) & \text{if } c(e) < k/2 \\ c(e) - k/2 & \text{otherwise} \end{cases}$$

If a run of the algorithm outputs the coloring c, then it must obtain the coloring c' at the end of the recursion step. In order to obtain the coloring c' at the end of the recursion step, it is necessary that for all $v \in V$ and $i \in \{1, \ldots, k/2\}$, the two arcs having their tails (resp. heads) at v and colored i in c', must separate during the splitting of the vertex v. Thus, the probability that the arcs having tails (resp. heads) at v get distributed correctly between v_0 and v_1 is $2^{k/2}/\binom{k}{k/2}$. The probability that the vertex v gets split correctly is $\left[2^{k/2}/\binom{k}{k/2}\right]^2$. Therefore, the probability that all n vertices get split correctly is $\left[2^{k/2}/\binom{k}{k/2}\right]^{2n}$, since the vertices are split independently.

The probability of obtaining c' after the recursive call, given that all vertices split correctly, is at most $f(2n, k/2)$, by induction. Thus, the probability that RandCycleCoverColoring outputs c is at most

$$\left[\frac{2^{k/2}}{\binom{k}{k/2}}\right]^{2n} \times f(2n, k/2) = \left[\frac{2^{k/2}}{\binom{k}{k/2}}\right]^{2n} \times \left[\frac{(k/2)^{k/2}}{((k/2)!)^2}\right]^{2n} = \left[\frac{k^k}{(k!)^2}\right]^n = f(n, k)$$

◀

Using the fact, $\ln(k!) \geq k \ln k - k$, arising from the Stirling's approximation, we have

$$f(n,k) = \left[\frac{k^k}{(k!)^2}\right]^n \leq \left[\frac{k^k}{(k/e)^{2k}}\right]^n = \left(\frac{e^2}{k}\right)^{kn} \tag{1}$$

We next bound from above the number of cycle covers with exactly r components.

▶ **Lemma 9.** *Let $G = (V, A)$ be a k-regular directed graph with n vertices (where k is not necessarily a power of 2). The number of cycle covers of G having r cycles is at most $\binom{n}{r}k^{n-r}$.*

Proof. Number the vertices of G arbitrarily. Consider a cycle cover $C \subseteq A$ of G which has r components, and let $(S_1, \ldots, S_r)$ be the partition of V induced by C, where $S_1, \ldots, S_r$ are sorted by the smallest numbered vertices that they contain. We associate the tuple $(|S_1|, \ldots, |S_r|)$ with C.

Given a tuple $(s_1, \ldots, s_r)$ such that $\sum_{i=1}^r s_i = n$, let us upper bound the number of cycle covers C of G that could be associated with this tuple. Since each cycle in G has length at least 2, we have each $s_i \geq 2$, and hence $r \leq n/2$. Let $(S_1, \ldots, S_r)$ be the partition induced by C, sorted by the smallest numbered vertices that they contain; $s_i = |S_i|$. Given $S_1, \ldots, S_{i-1}$, the smallest numbered vertex v_0 not in $S_1 \cup \cdots \cup S_{i-1}$ must be in S_i, and that must be the smallest numbered vertex in S_i too. Let the cycle containing v_0 in C be $(v_0, \ldots, v_{s_i-1})$ where $S_i = \{v_0, \ldots, v_{s_i-1}\}$. Then each v_j must be one of the k out-neighbors of v_{j-1}. Thus, given $S_1, \ldots, S_{i-1}$, the number of possibilities for S_i is at most k^{s_i-1}. Therefore, the number of cycle covers of G associated with the tuple $(s_1, \ldots, s_r)$ is at most $k^{\sum_{i=1}^r (s_i-1)} = k^{n-r}$.

Finally, by elementary counting, the number of tuples $(s_1, \ldots, s_r)$, for a fixed r, such that $\sum_{i=1}^r s_i = n$ and each $s_i \geq 2$, is $\binom{n-r-1}{r-1} < \binom{n}{r}$ for $r \leq n/2$. Thus, the number of cycle covers of G having r cycles is at most $\binom{n}{r}k^{n-r}$. ◀

Now we are ready to prove Theorem 7.

Proof of Theorem 7. Given a k-regular directed graph G with n vertices, let $t = \lfloor \gamma n / \ln k \rfloor$, where $\gamma > 2$ is a constant independent of n as well as k, which we will fix later. Call a cycle cover of G *bad* if it contains more than t components; else call it *good*. Call a cycle cover coloring $c : A \longrightarrow \{1, \ldots, k\}$ of G *bad* if for each i, the cycle cover $c^{-1}(i)$ is bad; else call it

good. We need to prove an upper bound on the probability of failure, that is, the probability that the random cycle cover coloring sampled by RandCycleCoverColoring is bad.

Since each cycle in G has length at least two, a cycle cover can contain at most $n/2$ components. Thus, if $t \geq n/2$, there is nothing to prove. So assume $t < n/2$. By Lemma 9, the number of bad cycle covers is at most

$$\sum_{r=t+1}^{n/2} \binom{n}{r} k^{n-r} \leq \left(\frac{n}{2} - t\right) \binom{n}{t} k^{n-t} \leq \frac{n}{2} \cdot 2^n \cdot k^{n-t}$$

where the first inequality follows from the fact that the function $r \longmapsto \binom{n}{r} k^{n-r}$ attains its maximum at $\lfloor \frac{n+1}{k+1} \rfloor < t$, and it is non-increasing in $\left[\lfloor \frac{n+1}{k+1} \rfloor, n\right]$. The number of bad cycle cover colorings is at most the number of ordered tuples of k bad cycle covers, which is at most

$$\left(\frac{n}{2}\right)^k 2^{nk} k^{k(n-t)} \leq \left(\frac{n}{2}\right)^k 2^{nk} k^{k\left(n - \frac{\gamma n}{\ln k} + 1\right)} = \left(\frac{n}{2}\right)^k 2^{nk} \left(\frac{k}{e^\gamma}\right)^{nk} k^k$$

Let c be the random cycle cover coloring output by the algorithm. By Lemma 8 and equation (1), the probability that c is bad is given by

$$\Pr[c \text{ is bad}] \leq \left(\frac{n}{2}\right)^k 2^{nk} \left(\frac{k}{e^\gamma}\right)^{nk} k^k \times \left(\frac{e^2}{k}\right)^{kn} = \left(\frac{2}{e^{\gamma-2}}\right)^{nk} \times \left(\frac{n}{2}\right)^k k^k$$

We take $\gamma = 2 + \ln 2 + \varepsilon$ so that $2/e^{\gamma-2} < 1$. This results in probability of failure decaying exponentially as n increases. ◀

4 The Deterministic Approximation Algorithm

The approach here is the similar to that of the randomized algorithm: find a small number of cycles in the graph covering a large number of vertices, and connect them using a spanning tree. The main difference is that while we construct a cycle cover in the previous algorithm, here we find a collection of vertex-disjoint cycles covering almost half the vertices. As before, we contract the cycles, and connect them and the uncovered vertices together with a spanning tree. Algorithm 2 essentially does a depth-first traversal, while repeatedly removing long cycles and vertices that cannot be fit in long cycles.

From the description, it is clear that this algorithm runs in time $O(nk)$, and that it finds cycles of length no less than $d = 2\sqrt{k}$. In order to derive the approximation ratio of our algorithm, we first need to bound from above the size of the set B returned by LongCycles. Let $m = |B|$.

▶ **Lemma 10.** $m \leq \frac{n(k-2)}{2(k-d+1)}$

Proof. Suppose the set B of vertices returned by the algorithm is $\{u_1, \ldots, u_m\}$, with the vertices added in the order $u_1, \ldots, u_m$. Consider the snapshot of the algorithm when the vertex u_i was added to B. At that time, vertices $u_1, \ldots, u_{i-1}$ were already removed from H and added to B, $u_{i+1}, \ldots, u_m$ were still present in H, and u_i was the last vertex in P. If $u \in \{u_{i+1}, \ldots, u_m\}$ is a neighbor of u_i, then u must be in P, otherwise, some neighbor of u_i would have been appended to P, rather than u_i getting removed from P. Further, the distance between u and u_i on P would be less than $d-1$, otherwise, a cycle would have been removed instead. Thus, the number of neighbors of u_i among $u_{i+1}, \ldots, u_m$ must be at most $d-2$. Therefore, the number of edges in the subgraph of G induced by B is less

Algorithm 2 LongCycles(G)

{INPUT: $G = (V, E)$, a k-regular n vertex directed graph; OUTPUT: A collection of cycles $\mathcal{C}$, each having length at least $2\sqrt{k}$, and a set B of vertices not in any cycle in $\mathcal{C}$.}
Initialize $H := G$, $\mathcal{C} = \emptyset$, $B := \emptyset$, $P := ()$, $d = 2\sqrt{k}$.
{P always remains a path in H.}
while H is nonempty **do**
  if P is empty **then**
    Add an arbitrary vertex of H to P.
  else
    {Suppose $P = (v_1, \ldots, v_t)$ with $t > 0$.}
    if v_t has a neighbor u in H outside P **then**
      Append u to P.
    else if $t \geq d$ and v_t has a neighbor v_s in P for $s \leq t - d + 1$ **then**
      Remove the vertices $v_s, v_{s+1}, \ldots, v_{t-1}, v_t$ from P and H; add this cycle to $\mathcal{C}$.
    else
      Remove v_t from P and H, and add it to B.
    end if
  end if
end while
Return $\mathcal{C}, B$.

than $(d-2)m$. As a consequence, the number of edges in G between B and $V \setminus B$ is at least $km - 2(d-2)m = (k - 2d + 4)m$.

Next, the number of vertices in $V \setminus B$ is $n - m$ and this is exactly the set of vertices covered by cycles in $\mathcal{C}$. For each vertex in $V \setminus B$, at most $k - 2$ of the k edges incident on it have their other endpoint in B. Thus, the number of edges between B and $V \setminus B$ is at most $(n - m)(k - 2)$. Hence $(k - 2d + 4)m \leq (n - m)(k - 2)$, which implies $m \leq \frac{n(k-2)}{2(k-d+1)}$. ◀

The above lemma implies that almost half of the vertices are covered by cycles in $\mathcal{C}$. We next use it to prove the approximation ratio.

▶ **Theorem 11.** *Consider the algorithm for finding a TSP tour, which runs* LongCycles *on the input graph, and connects the cycles in $\mathcal{C}$ using two copies of a spanning tree of the graph obtained by contracting the cycles. The approximation ratio of this algorithm is $\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)$.*

Proof. Since the vertex-disjoint cycles in $\mathcal{C}$ cover $n - m$ vertices, and each cycle contains at least d vertices, the number of cycles in $\mathcal{C}$ is at most $(n - m)/d$, and hence, the number of components to be connected using a spanning tree is at most $(n - m)/d + m$. The TSP tour that the algorithm constructs consists of the cycles in $\mathcal{C}$, and two copies of a spanning tree in the graph obtained by contracting the cycles. The former contributes $n - m$ edges, while the

latter contributes at most $2(n-m)/d + 2m - 2$ edges. Thus, the cost of the tour is at most

$$\begin{aligned} n - m + \frac{2(n-m)}{d} + 2m - 2 &= n\left(1 + \frac{2}{d}\right) + m\left(1 - \frac{2}{d}\right) - 2 \\ &\leq n\left(1 + \frac{2}{d}\right) + \frac{n(k-2)}{2(k-d+1)}\left(1 - \frac{2}{d}\right) \\ &\leq n\left(1 + \frac{2}{d} + \frac{k-2}{2(k-d+1)}\right) \\ &= n\left(\frac{3}{2} + \frac{2}{d} + \frac{d-3}{2(k-d+1)}\right) \end{aligned}$$

where we have used Lemma 10 for the first inequality. For $d = \Theta\left(\sqrt{k}\right)$, the cost of the tour turns out to be $n\left(\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)\right)$. Thus, the algorithm achieves a $\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)$ factor approximation. ◀

5 Concluding Remarks

Vishnoi's algorithm as well as both of our algorithms work only on regular graphs. Extending these to work on a larger class of graphs, with weaker assumptions about the vertex degrees, is an interesting problem, and will involve new techniques. Indeed, Feige et al. [10] have initiated research on this front. We used the number of vertices as a lower bound on the cost of the optimal TSP tour. Extending to a larger class of graphs will require a tighter lower bound, and the cost of the Held-Karp relaxation is one such candidate. Even for regular graphs, we do not know a hardness of approximation result, as a function of the degree k. Indeed improving the approximation factor to $1 + O(1/k)$ cannot be ruled out.

We would like to see whether our algorithm can be derandomized to get a $(1 + o(1))$-approximation, possibly with some loss in the running time. We strongly feel that the following related avenues are worth exploring: first, to determine the best approximation ratio that can be achieved by deterministic algorithms for the RegGraphTSP, and second, to determine the best approximation ratio that can be achieved by linear time deterministic algorithms.

Finally, we feel it would be interesting to use edge coloring ideas to come up with fast sampling procedures which give better guarantee on the resulting sampling distribution on matchings, than ours.

Acknowledgments. The authors thank Nisheeth Vishnoi and Parikshit Gopalan for some initial discussions. The authors also thank Ayush Choure for his substantial contribution to this paper.

References

1 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

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 Yair Bartal and Lee-Ad Gottlieb. A linear time approximation scheme for euclidean TSP. In *FOCS*, pages 698–706. IEEE, 2013.

4 Ivona Bezáková, Daniel Stefankovic, Vijay V. Vazirani, and Eric Vigoda. Accelerating simulated annealing for the permanent and combinatorial counting problems. *SIAM J. Comput.*, 37(5):1429–1454, 2008.

5 Hans-Joachim Böckenhauer and Sebastian Seibert. Improved lower bounds on the approximability of the traveling salesman problem. *ITA*, 34(3):213–255, 2000.

6 Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. The traveling salesman problem on cubic and subcubic graphs. *Math. Program.*, 144(1-2):227–245, 2014.

7 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

8 Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001.

9 Lars Engebretsen. An explicit lower bound for TSP with distances one and two. *Algorithmica*, 35(4):301–318, 2003.

10 Uriel Feige, R. Ravi, and Mohit Singh. Short tours through large linear forests. In *IPCO*, volume 8494 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2014.

11 Harold N. Gabow and Oded Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comput.*, 11(1):117–129, 1982.

12 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *FOCS*, pages 550–559. IEEE, 2011.

13 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

14 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

15 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

16 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. In *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 568–578. Springer, 2013.

17 Michael Lampis. Improved inapproximability for TSP. In *APPROX-RANDOM*, volume 7408 of *Lecture Notes in Computer Science*, pages 243–253. Springer, 2012.

18 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.

19 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *FOCS*, pages 560–569. IEEE, 2011.

20 Marcin Mucha. 13/9-approximation for graphic TSP. In *STACS*, volume 14 of *LIPIcs*, pages 30–41. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012.

21 Christos H. Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

22 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

23 Satish Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *STOC*, pages 540–550. ACM, 1998.

24 Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.

25 András Sebö and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

26 Luca Trevisan. Inapproximability of combinatorial optimization problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004(065), 2004.

27 Nisheeth K. Vishnoi. A permanent approach to the traveling salesman problem. In *FOCS*, pages 76–80. IEEE, 2012.

On Weighted Bipartite Edge Coloring

Arindam Khan[*1] **and Mohit Singh**[2]

1 **Georgia Institute of Technology**
Atlanta, USA
akhan67@gatech.edu
2 **Microsoft Research**
Redmond, USA
mohits@microsoft.com

Abstract

We study WEIGHTED BIPARTITE EDGE COLORING problem, which is a generalization of two classical problems: BIN PACKING and EDGE COLORING. This problem has been inspired from the study of Clos networks in multirate switching environment in communication networks. In WEIGHTED BIPARTITE EDGE COLORING problem, we are given an edge-weighted bipartite multigraph $G = (V, E)$ with weights $w : E \to [0, 1]$. The goal is to find a *proper weighted coloring* of the edges with as few colors as possible. An edge coloring of the weighted graph is called a *proper weighted coloring* if the sum of the weights of the edges incident to a vertex of any color is at most one. Chung and Ross conjectured $2m - 1$ colors are sufficient for a proper weighted coloring, where m denotes the minimum number of unit sized bins needed to pack the weights of all edges incident at any vertex. We give an algorithm that returns a coloring with at most $\lceil 2.2223m \rceil$ colors improving on the previous result of $\frac{9m}{4}$ by Feige and Singh. Our algorithm is purely combinatorial and combines the König's theorem for edge coloring bipartite graphs and first-fit decreasing heuristic for bin packing. However, our analysis uses *configuration linear program* for the bin packing problem to give the improved result.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Edge coloring, Bin packing, Clos networks, Approximation algorithms, Graph algorithms.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.136

1 Introduction

Clos networks were introduced by Clos [3] in the context of designing interconnection networks with small number of links to route multiple simultaneous connection requests such as telephone calls. Since then it has found various applications in data communications and parallel computing systems [1, 11]. The symmetric 3-stage Clos network is generally considered to be the most basic multistage interconnection network. Let $C(m, \mu, r)$ denote a symmetric 3-stage Clos network, where the input (first) stage consists of r crossbars (switches) of size $m \times \mu$, the center (second) stage consists of μ crossbars of size $r \times r$ and the output (third) stage consists of r crossbars of size $\mu \times m$. Moreover, there exists one link between every center switch and each of the r input or output switch. No link exists between other pair of switches.

* A part of this work was done when the author was interning at Microsoft Research. The author also thanks NSF EAGER award grants CCF-1415496 and CCF-1415498.

A request frame is a collection of connection requests between inlets and outlets in the network such that each inlet or outlet is associated with at most one request. A request frame is *routable* if all requests are routed through a middle switch such that no two requests share same link. An interconnection network is called to be *rearrangeably nonblocking* if all request frames are routable. In the classic switching environment all connection requests fully use a link and all have same bandwidth. However in modern networks, different requests might have different bandwidths (due to wide range of traffic such as voice, video, facsimile etc.) and may be combined in a given link if the combined request does not exceed the link capacity. In this *multirate* setting, a connection request is a triple (i, j, w) where i, j, w are inlet, outlet and demand of the connection respectively, and all links have capacity one. Here a request frame is a collection of connection requests between inlets and outlets in the network such that the total weight of all requests in the frame for any particular inlet or outlet is at most one. The central question in 3-stage Clos networks is finding the minimum number of crossbars μ $(= \mu(m, r))$ in the middle stage such that all request frames are routable. It is more interesting to obtain bounds independent of r.

Nonblocking rearrangeable properties of 3-stage Clos network $C(m, \mu, r)$ can be translated to the following graph theoretic problem. Formally, in WEIGHTED BIPARTITE EDGE COLORING problem, we are given an edge-weighted bipartite (multi)-graph $G := (V, E)$ with bipartitions A, B $(|A| = |B| = r)$ and edge weights $w : E \to [0, 1]$. Let w_e denote the weight of edge $e \in E$. The goal is to obtain a *proper weighted coloring* of all edges with minimum number of colors. An edge coloring of the weighted bipartite graph is called a *proper weighted coloring* if the sum of the same color edges incident to a vertex is at most one for any color and any vertex. Here the sets A and B correspond to the input and output switches, edge (u, v) corresponds to a request between input switch u and output switch v. A routable request frame translates into the condition that weights of all incident edges to any vertex can be proper weighted colored using m colors (or packed into m unit sized bins) and the switches in the middle stage correspond to the colors (or bins). We refer the reader to Correa and Goemans [5] for detailed discussion of this reduction.

The weighted bipartite edge coloring problem naturally generalizes two classically studied problems, the EDGE COLORING PROBLEM and the BIN PACKING PROBLEM. If all edge weights are one, this problem reduces to the classical edge coloring problem. On the other hand, if there is only one vertex in each partition in the bipartite graph, this problem reduces to the bin packing problem.

Now let us introduce some notation. Let $\chi'_w(G)$ denote the minimum number of colors needed to obtain a proper weighted coloring of G. Let $m, r \in \mathbb{Z}^+$, and $\mu(m, r) = max_G \chi'_w(G)$ where the maximum is taken over all bipartite graphs $G = (A \cup B, E)$ with $|A| = |B| = r$ and where m is the maximum over all the vertices of the number of unit-sized bins needed to pack the weights of incident edges. Chung and Ross [2] made the following conjecture.

▶ **Conjecture 1.** *Given an instance of the* WEIGHTED BIPARTITE EDGE COLORING *problem, there is a proper weighted coloring using at most* $2m-1$ *colors where* m *denotes the maximum over all the vertices of the number of unit-sized bins needed to pack the weights of edges incident at the vertex. In other words,* $\mu(m, r) \leq 2m - 1$.

There has been a series of results achieving weaker bounds on $\mu(m, r)$ (see related works for details), the current best bound by Feige and Singh [7] shows that $\mu(m, r) \leq 2.25m$.

1.1 Our results and techniques.

Our main result is to make progress towards resolution of Conjecture 1 by showing $\mu(m, r) \leq \frac{20m}{9} + o(m)$.

▶ **Theorem 2.** *There is a polynomial time algorithm for the* WEIGHTED BIPARTITE EDGE COLORING *problem which returns a proper weighted coloring using at most* $\lceil 2.2223m \rceil$ *colors where* m *denotes the maximum over all the vertices of the number of unit-sized bins needed to pack the weights of incident edges.*

In our algorithm and analysis, we exploit that WEIGHTED BIPARTITE EDGE COLORING problem displays features of the classical edge coloring problem as well as the bin packing problem. Our algorithm starts by decomposing the *heavy* weight edges into matchings by applying König's theorem to find an edge coloring of the subgraph induced by these edges. For the light weight edges, we employ the *first-fit decreasing* heuristic where we consider the remaining edges in decreasing order of weight and give them the first available color. The detailed algorithm is given in Figure 1 and builds on the algorithm by Feige and Singh [7].

Our work diverges from previous results on this problem in the analysis of this simple combinatorial algorithm. We employ strong mathematical formulations for the bin packing problem; in particular, we use the *configuration linear program (LP)* for the bin packing problem. This linear program has been used to design the best approximation algorithm for the bin packing problem [12, 17, 9]. In our work, we use it as follows. We show that if the algorithm is not able to a color an edge (u, v), then the edges incident at u or v cannot be packed in m bins as promised. To show this, we formulate the configuration linear program for the two bin packing problems, one induced by edges incident at u and the other induced by edges incident at v. We then construct feasible dual solutions to one of these linear programs of value more than m. Appealing to linear programming duality, it implies that the optimal primal value, and therefore the optimal bin packing number, is more than m for at least one of the programs, giving us the desired contradiction. While the weights on the edges incident at u (or v) can be arbitrary reals between 0 and 1, we group the items according to weight classes and how our algorithm colors these edges. This allows us to reduce the number of item types, reducing the complexity of the configuration LP and makes it easier to analyze. While the grouping according to weight classes is natural in bin packing algorithms; the grouping based on the output of our algorithm helps us relate the fact that the edge (u, v) could not be colored by our algorithm to the bin packing bound at u and v. We mention two additional extensions of our techniques to the problem. Firstly, a more careful and detailed analysis (based on computer search) can improve the bounds slightly showing the current bound is not tight. Secondly, our analysis can also be extended to show $\lceil 2.2m \rceil$ colors are sufficient when all edge weights are more than $1/4$.

1.2 Related Works

Edge-coloring problem has been one of the central problems in graph theory and discrete mathematics since its appearance in 1880 [21] in relation with the *four-color problem*. The chromatic index of a graph is the number of colors required to color the edges of the graph such that no two adjacent edges have the same color. Three classical results on edge coloring are König's theorem [14] for coloring a bipartite graph with Δ colors, Vizing's theorem [22] for coloring any simple graph with $\Delta + 1$ colors and Shanon's theorem [18] for coloring any multigraph with at most $3\Delta/2$ colors where Δ is the maximum degree of the graph. Though one can find optimal edge coloring for a bipartite graph in polynomial time using König's

1. $F \leftarrow \emptyset$, $t \leftarrow 2.2223$.
2. Include edges with weight $> \gamma = \frac{1}{10}$ in F in nonincreasing order of weight maintaining the property that $deg_F(v) \leq \lceil tm \rceil$ for all $v \in V$.
3. Decompose F into $r = \lceil tm \rceil$ matchings $M_1, \ldots, M_r$ and color them using colors $1, \ldots, r$. Let $F_i \leftarrow M_i$ for each $1 \leq i \leq r$.
4. Add remaining edges in nonincreasing order of weight to any of the F_i's maintaining that weighted degree of each color at each vertex is at most one, i.e., $\sum_{e \in \delta(v) \cap F_i} w_e \leq 1$ for each $v \in V$ and $1 \leq i \leq r$.

Figure 1 Algorithm for Edge Coloring Weighted Bipartite Graphs.

theorem, Holyer [10] showed that it is NP-hard even to decide whether the chromatic index of a cubic graph is 3 or 4. We refer the readers to [20] for a survey on edge coloring.

On the other hand classical bin packing problem is NP-hard and has been studied extensively from the classical work of Garey et al. [8]. The problem finds numerous applications in scheduling, logistics, layout design and other resource allocation problems. The present best polynomial time algorithm is due to Hoberg and Rothvoß [9] based on rounding of configuration LP using connections to discrepancy [17] and achieves a logarithmic additive error. However only known hardness bound is Opt + 1 assuming P $\neq$ NP. We refer the readers to [4] for a survey on the current literature for bin packing.

Now we review the literature related to weighted bipartite edge coloring. First, we introduce some more notation. When the weight function $w : E \rightarrow I$ is restricted to a subinterval $I \subseteq [0, 1]$, then we denote the minimum number of colors by $\mu_I(m, r)$. Slepian [19] showed that $\mu_{[1,1]}(m, r) = m$ using König's theorem. Melen and Turner [15] showed that $\mu_{[0,B]}(m, r) \leq \frac{m}{1-B}$ for $B \leq 1$. In particular, $\mu_{[0,1/2]}(m, r) \leq 2m - 1$. There has been a series of work improving the bounds for $\mu(m, r)$ [2, 6, 16, 5]. The best known lower bound for $\mu(m, r)$ is 5/4 due to Ngo and Vu [16]. Correa and Goemans introduced a novel graph decomposition result and perfect packing of an associated continuous bin packing instance to show $\mu(m, r) \leq 2.5480m + o(m)$. The present best algorithm is due to Feige and Singh [7] who showed $\mu(m, r) \leq \frac{9m}{4}$. Their result holds even if m is the maximum over all the vertices of the total weight of edges incident at the vertex. For other related results, see [7].

2 Edge Coloring Weighted Bipartite Graphs

In this section we present our main result and prove Theorem 2.

▶ **Theorem 3** (Restatement of Theorem 2). *There is a polynomial time algorithm for the* WEIGHTED BIPARTITE EDGE COLORING *problem which returns a proper weighted coloring using at most* $\lceil 2.2223m \rceil$ *colors where* m *denotes the maximum over all the vertices of the number of unit-sized bins needed to pack the weights of incident edges.*

Our complete algorithm for edge-coloring weighted bipartite graphs is given in Figure 1. In the algorithm, we set a threshold $\gamma = \frac{1}{10}$ and consider the subgraph induced by edges with weights more than γ and apply a combination of König's Theorem and a greedy algorithm with $\lceil tm \rceil$ colors where $t = 2.2223 > 20/9$. The remaining edges of weights at most γ are then added greedily.

Analysis

Now we prove Theorem 2. Though the algorithm is purely combinatorial, the analysis uses configuration LP and other techniques from bin packing to prove the correctness of the algorithm. First, we state the König's Theorem since we use it as a subroutine in our algorithm to ensure a decomposition of F into $\lceil tm \rceil$ matchings.

▶ **Theorem 4** ([14]). *Given a bipartite graph $G = (V, E)$, there exists a coloring of edges with $\Delta = max_{v \in V} deg_E(v)$ colors such that all edges incident at a common vertex receive a distinct color. Moreover, such a coloring can be found in polynomial time.*

The following lemma from Correa and Goemans [5] (it was also implicit in [6]) ensures that if the algorithm succeeds in coloring all edges of weight at least γ, the greedy coloring will be able to color the remaining edges of weight at most γ.

▶ **Lemma 5** ([5, 6]). *Consider a bipartite weighted graph $G = (V, E)$ with a coloring of all edges of weight $> \gamma$ using at least $\frac{2m}{1-\gamma}$ colors for some $\gamma > 0$. Then the greedy coloring will succeed in coloring the edges with weight at most γ without any additional colors.*

In our setting, we have $\gamma = \frac{1}{10}$ and the number of colors is $\leq \frac{20}{9}m = \frac{2m}{1-\frac{1}{10}}$ and thus Lemma 5 applies. Hence, it suffices to show the algorithm is able to color all edges with weight $> \frac{1}{10}$ using $\lceil tm \rceil$ colors as the remaining smaller edges can be colored greedily. Thus, w.l.o.g, we assume that the graph has no edges of weight $\leq \frac{1}{10}$ and prove the following lemma.

▶ **Lemma 6.** *If all edges have weight more than $\frac{1}{10}$ and $t = 2.2223$ $(> 20/9)$ then the algorithm in Figure 1 returns a coloring of all edges using $\lceil tm \rceil$ colors such that the weighted degree of each color at each vertex is at most one, i.e., $\sum_{e \in \delta(v) \cap F_i} w_e \leq 1$.*

Proof. Suppose for the sake of contradiction, the algorithm is not able to color all edges. Let $e := (u, v)$ be the first edge that cannot be colored by any color in Step (3) or Step (4) of the algorithm. Let the weight of edge e, w_e, be α. Moreover, when e is considered in Step (2), degree of either u or v is already $\lceil tm \rceil$ else we would have included e in F. Without loss of generality let that vertex be u, i.e., $deg_F(u) = \lceil tm \rceil$. Now we characterize the colors F_i of edges incident at u and consider the edges incident at u and v to get a series of inequalities. Thereafter, we show that $\alpha \leq 1/3$ and use these inequalities to arrive at a contradiction.

For each color $1 \leq i \leq \lceil tm \rceil$, either $\sum_{f \in \delta(v) \cap F_i} w_f > 1 - \alpha$ or $\sum_{f \in \delta(u) \cap F_i} w_f > 1 - \alpha$, else we can color e in Step (4). Let $H_v = \{i | \sum_{f \in \delta(v) \cap F_i} w_f > 1 - \alpha\}$, $\beta m = |H_v|$. Now for each color $i \notin H_v$, we have $\sum_{f \in \delta(u) \cap F_i} w_f > 1 - \alpha$. Moreover, $deg_F(u) = \lceil tm \rceil$ and for all edge $f \in \delta(u)$, we have $w_f \geq \alpha$ as the edges were considered in nonincreasing order of weight. Hence, for each color $1 \leq i \leq \lceil tm \rceil$, there is an edge incident at u colored with color i with weight at least α. Let us call a color i *tight* at u if $\sum_{f \in \delta(u) \cap F_i} w_f > (1 - \alpha)$ and a color i *open* at u if $\sum_{f \in \delta(u) \cap F_i} w_f \in [\alpha, 1 - \alpha]$. Let τ be the number of tight colors at u and θ be the number of open colors at u. Thus we have,

$$\tau \geq (t - \beta)m \tag{1}$$

$$\theta = (tm - \tau) \leq \beta m \tag{2}$$

Now consider all edges incident on v. We get, $m > \beta m(1 - \alpha)$

$$\Rightarrow \quad 1 > \beta(1 - \alpha) \tag{3}$$

Similarly considering all edges incident on u, we get: $m > (tm - \beta m)(1 - \alpha) + (\beta m)\alpha$

$$\Rightarrow \quad 1 > t(1 - \alpha) + \beta(2\alpha - 1) \tag{4}$$

Now a unit sized bin can contain at most two items with weight $> 1/3$. As all edges incident to a vertex can be packed into m unit sized bins, there can be at most $2m$ edges incident to a vertex with weight $> 1/3$. Since $t > 2$, we get that all edges with weight more than $\frac{1}{3}$ must have been included in F in Step (2). Thus $\alpha \leq 1/3$.

Thus we get from (3): $$\beta < 1/(1-\alpha) \leq 3/2 \tag{5}$$

Now there are two cases:

Case A: $\alpha \leq 1/4$. Consider the RHS of (4): $t(1-\alpha) + \beta(2\alpha - 1)$. Now,

$$\begin{aligned} t(1-\alpha) + \beta(2\alpha-1) - 1 &> t(1-\alpha) - \frac{(1-2\alpha)}{(1-\alpha)} - 1, \quad \text{[From (3)]} \\ &\geq \frac{20(1-\alpha)^2 - 9(1-2\alpha) - 9(1-\alpha)}{9(1-\alpha)} \quad [\because t > 20/9] \\ &\geq \frac{(20\alpha^2 - 13\alpha + 2)}{9(1-\alpha)} = \frac{(4\alpha-1)(5\alpha-2)}{9(1-\alpha)} \geq 0, \text{as } \alpha \leq 1/4 \end{aligned}$$

Thus $t(1-\alpha) + \beta(2\alpha - 1) > 1$, which contradicts (4).

Case B: $1/4 < \alpha \leq 1/3$. In this case, we will show in Lemma 7 that if $\beta \leq 13/9$, then all edges incident at u can not be packed in m bins. On the other hand, in Lemma 14 we show that if $\beta > 13/9$, then all edges incident at v can not be packed in m bins.

This two facts together give us the desired contradiction.

▶ **Lemma 7.** *If $\beta \leq 13/9$, then edges incident at u can not be packed in m bins.*

Proof. To give a lower bound on the number of bins required, we will consider a relaxation to the bin packing problem for edges incident at vertex u and show that the optimal value of the relaxation, and thus the optimal number of bins required, is greater than m. The lower bound will be exhibited by constructing a feasible dual solution to the relaxation to the bin packing problem.

Since $deg_F(u) = \lceil tm \rceil$ when edge e was considered in Step (2) of the algorithm and not included in F, we have that all edges incident at u in F have weight at least the weight of e. Moreover, edges are considered in the decreasing order of weight in Step (4), the weight of all edges incident at u when e is considered in Step (4) is $\geq w_e$. We restrict our attention to these edges incident at u with weight $\geq \alpha$ and show that they cannot be packed in m unit sized bins. Let us divide these edges incident at u into three size classes.

- Large $L := \{f \in \delta(u) : w_f \in (1/2, 1]\}$.
- Medium $M := \{f \in \delta(u) : w_f \in (1/3, 1/2]\}$.
- Small $S := \{f \in \delta(u) : w_f \in [\alpha, 1/3]\}$.

First we have the following observation.

▶ **Observation 8.** *In any bin packing solution, in any bin there can be at most one item from L, two items from $L \cup M$ and three items from $L \cup M \cup S$.*

Now consider the following two simple claims.

▶ **Claim 9.** *Edges in $L \cup M$ are included in Step (2) of Algorithm 1 and are a subset of F.*

Proof. If we are unable to add an edge f in Step (2), it means one of its endpoints have $\lceil tm \rceil > 2m$ edges with weight $\geq w_f$. Since all edges incident at any vertex $v \in V$ can be packed in m bins, there are at most $2m$ edges incident at it with weight more than $\frac{1}{3}$. Thus all edges of weight more than $\frac{1}{3}$, i.e., all edges in $L \cup M$ must be included in F in Step (2) of the algorithm. ◀

▶ **Claim 10.** *For any color i, there is at most one edge in $L \cup M$ with color i.*

Proof. As all edges in $L \cup M$ must be included in F in Step (2) of the algorithm. In Step (3) of the algorithm, we include at most one edge of F incident at any vertex in each F_i. Thus each color class obtains at most one edge incident at each vertex from F and therefore, from $L \cup M$. ◀

Using the observation and the above claim, we itemize the configuration of each of the tight colors depending on the size of edges with that color. Note that tight colors must have weight $> 1 - \alpha \geq 1 - 1/3 = 2/3$.

1. If the tight color has a single edge f. Then we have that $w_f > \frac{2}{3}$ and only possibility is $i)(L)$; Here by (L), we denote that the bin contains only one item and that item is an item from the set L.
2. If the tight color contains exactly two edges. Here (S, S) is not tight as the total weight of edges in such a bin is $\leq 2/3$. So, the bin can contain at most one item from S. On the other hand, the bin can contain at most one item from $L \cup M$ from Claim 10. Thus the possible size types of these edges are $ii)(L, S); iii)(M, S)$; As above, by (L, S) we denote that the bin contains only two items: exactly one item from set L and exactly one item from S.
3. If the tight color contains three edges. The bin can contain at most one item from $L \cup M$ from Claim 10. However if it contains one L item, sum of weights of three items exceeds one. Thus the only possible size types of these edges are $iv)(M, S, S); v)(S, S, S)$.

Now consider the following LP: $LP_{bin}(u)$:

$$min \sum_{i=1}^{5} y_i$$

$$x_1 + x_2 + x_3 + x_4 + x_5 \geq \tau \tag{6}$$

$$y_1 + y_2 \geq x_1 + x_2 + z_1 \tag{7}$$

$$y_1 + 2y_3 + y_4 \geq x_3 + x_4 + z_2 \tag{8}$$

$$y_2 + y_3 + 2y_4 + 3y_5 \geq x_2 + x_3 + 2x_4 + 3x_5 + z_3 \tag{9}$$

$$z_1 + z_2 + z_3 \geq \theta \tag{10}$$

$$x_j, y_k, z_l \geq 0 \quad \forall j \in [5], k \in [5], l \in [3] \tag{11}$$

▶ **Lemma 11.** *The optimal number of unit sized bins needed to pack all edges incident at u is at least the optimum value of $LP_{bin}(u)$.*

Proof. Given a feasible packing of edges incident at u in at most m unit sized bins, we construct a feasible solution $(\bar{x}, \bar{y}, \bar{z})$ to the linear programming relaxation whose objective is at most the number of unit sized bins needed in the packing. In the feasible solution $(\bar{x}, \bar{y}, \bar{z})$, the variables $\bar{x}$ and $\bar{z}$ are constructed using the coloring given by the algorithm. The variables $\bar{y}$ are constructed using the optimal bin packing.

We first define the variables $\bar{x}$. Let $\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5$ be the number of tight colors at u of type $(L), (L, S), (M, S), (M, S, S), (S, S, S)$, respectively. Since the coloring of the edges incident at u is one of the five types described above and there are at least τ tight colors, we have that $\sum_{i=1}^{5} \bar{x}_i \geq \tau$ and thus the solution satisfies constraint (6). Now we define the variables $\bar{z}_1, \bar{z}_2, \bar{z}_3$ to be the number of items in open colors from L, M and S respectively. There are θ open colors. Each open color contains at least one item $L \cup M \cup S$. Thus, $\bar{z}_1 + \bar{z}_2 + \bar{z}_3 \geq \theta$ and thus the solution satisfies constraint (10).

To construct the solution $\bar{y}$, we will group the bins in the optimal bin packing solutions depending on the subset of items present in them into five classes and the number of bins in each class will define the variables $\bar{y}$. The constraints (7)-(9) will correspond to making sure that the optimal bin packing solution has appropriate number of items of each size type.

Now let us characterize the possible bin configurations to explain constraints (7)-(9).

▶ **Claim 12.** *Consider any feasible bin packing of edges incident at u restricted to edges in $L \cup M \cup S$. Then each bin must contain items which correspond to a subset of one of the following 5 configurations or subsets of these configurations.*

$C_1 : (L, M)$ $C_2 : (L, S)$ $C_3 : (M, M, S)$
$C_4 : (M, S, S)$ $C_5 : (S, S, S)$

Proof. Observe that in any bin there can be at most one item from L, two items from $L \cup M$ and three items from $L \cup M \cup S$. Now let us consider two cases.
1. If the bin contains an item from L. In this case, the bin can not contain three items as the sum of their weights exceeds one. So, it can contain at most one item from L and one item from $M \cup S$. Thus C_1 and C_2 cover such two cases.
2. If the bin does not contain any item from L. In this case, the bin can contain three items from $M \cup S$ and at most two of these items can be from M. Thus C_3, C_4 and C_5 cover such possibilities. ◀

We map each configuration in the optimal bin packing solution to one of types C_i where the configuration is either C_i or its subset. Let $\bar{y}_i$ denote the number of bins mapped to type C_i for each $1 \leq i \leq 5$. We now count the number of items of each type to show feasibility of the constraints of the linear program.

Constraint (7). Items of type L equal $\bar{x}_1 + \bar{x}_2 + \bar{z}_1$ and can only be contained in configuration C_1 and C_2. Thus we have $\bar{y}_1 + \bar{y}_2 \geq \bar{x}_1 + \bar{x}_2 + \bar{z}_1$ satisfying constraint (7).
Constraint (8). Items of type M equal $\bar{x}_3 + \bar{x}_4 + \bar{z}_2$ and are contained once in configurations C_1, C_4 and twice in configuration C_3. Thus we have $\bar{y}_1 + 2\bar{y}_3 + \bar{y}_4 \geq \bar{x}_3 + \bar{x}_4 + \bar{z}_2$ satisfying constraint (8).
Constraint (9). Items of type S equal $\bar{x}_2 + \bar{x}_3 + 2\bar{x}_4 + 3\bar{x}_5 + \bar{z}_3$ and occur once in configurations C_2, C_3, twice in configurations C_4 and thrice in C_5. Thus, we have $\bar{y}_2 + \bar{y}_3 + 2\bar{y}_4 + 3\bar{y}_5 \geq \bar{x}_2 + \bar{x}_3 + 2\bar{x}_4 + 3\bar{x}_5 + \bar{z}_3$ showing feasibility of constraint (9).
This implies that $(\bar{x}, \bar{y}, \bar{z})$ is a feasible solution to LP_{bin} and its objective equals the number of bins needed to pack the edges incident at u in unit sized bins. Thus we have the lemma. ◀

We now show a contradiction by showing the optimal value of the $LP_{bin}(u)$ is more than m.

▶ **Lemma 13.** *The optimal solution to $LP_{bin}(u)$ is strictly more than m.*

Proof. We prove this by considering the dual linear program of the $LP_{bin}(u)$. Since every feasible solution to the dual LP gives a lower bound on the objective of the primal $LP_{bin}(u)$, it is enough to exhibit a feasible dual solution of objective strictly more than m to prove the lemma. Now the dual of the LP_{bin} is given in the next page.

A feasible dual solution is: $v_1 = \frac{2}{3}, v_2 = \frac{2}{3}, v_3 = \frac{1}{3}, v_4 = \frac{1}{3}, v_5 = \frac{1}{3}$.

$$
\begin{aligned}
&max \quad \tau \cdot v_1 + \theta \cdot v_5 \\
&\text{Subject to:} \\
&v_1 - v_2 \leq 0, && v_1 - v_2 - v_4 \leq 0, \\
&v_1 - v_3 - v_4 \leq 0, && v_1 - v_3 - 2v_4 \leq 0, \\
&v_1 - 3v_4 \leq 0, && v_2 + v_3 \leq 1, \\
&v_2 + v_4 \leq 1, && 2v_3 + v_4 \leq 1, \\
&v_3 + 2v_4 \leq 1, && 3v_4 \leq 1, \\
&v_5 - v_2 \leq 0, && v_5 - v_3 \leq 0, \\
&v_5 - v_4 \leq 0, && v_i \geq 0 \quad \forall i \in [5]
\end{aligned}
$$

Thus dual optima $\geq \frac{2\tau}{3} + \frac{\theta}{3}$ and we need at least these many colors to color items in τ tight colors and θ open colors. Using the fact that $\theta = tm - \tau, \tau \geq (t-\beta)m$ and $t > \frac{20}{9}m, \beta \leq \frac{13}{9}$, we obtain that the number bins required to pack all items incident on u is:

$$
\begin{aligned}
&\geq \quad \tau \cdot v_1 + \theta \cdot v_4 \quad \geq \quad \frac{2}{3}\tau + \frac{1}{3}(tm - \tau) \quad = \quad \frac{1}{3}\tau + \frac{1}{3}(tm) \\
&\geq \quad \frac{1}{3}(t-\beta)m + \frac{1}{3}(tm) \quad \geq \quad \frac{2t}{3}m - \frac{\beta}{3}m \quad > \quad m(\frac{2}{3} \cdot \frac{20}{9} - \frac{1}{3} \cdot \frac{13}{9}) \quad = \quad m
\end{aligned}
$$

Thus the number bins required to pack all items incident on u is strictly greater than m. This is a contradiction. ◀

This concludes the proof of Lemma 7. ◀

▶ **Lemma 14.** *If $\beta > 13/9$, then edges incident at v can not be packed in m bins.*

Proof. Similar to the previous lemma, to give a lower bound on the number of bins required, we will consider a relaxation to the bin packing problem for edges incident at vertex v and show that the optimal value of the relaxation, and thus the optimal number of bins required, is greater than m. Again, the lower bound will be exhibited by constructing a feasible dual solution to the relaxation to the bin packing problem.

As $\beta(1-\alpha) < 1$, we get,

$$\alpha > 1 - 1/\beta \geq 4/13 > 0.3. \tag{12}$$

Let us call a color i *tight* at v if $\sum_{f \in \delta(v) \cap F_i} w_f > (1-\alpha)$. Now consider any tight color $\mathcal{B}$ at v. At most one edge f in $\mathcal{B}$ was colored in Step (3) of the algorithm and remaining edges (if any) in $\mathcal{B}$ were colored in Step (4) of the algorithm. Now, w_f can be smaller than w_e as it might be the case that when e was considered in Step (2) then already degree of other endpoint u was $\lceil tm \rceil$. However, edges are considered in the nonincreasing order of weight in Step (4), thus the weight of all edges incident at v when e is considered in Step (4) is also $\geq w_e$. Thus, all the remaining edges (if any) in $\mathcal{B}$ that were colored in Step (4) of the algorithm have weight more than α.

We restrict our attention to the edges at tight colors at v and show that if $\beta > \frac{13}{9}$ they cannot be packed in m unit sized bins. Let us divide these edges incident at u into four size classes.

- Large $L := \{f \in \delta(v) : w_f \in (1/2, 1]\}$.
- Medium $M := \{f \in \delta(v) : w_f \in (1/3, 1/2]\}$.
- Small $S := \{f \in \delta(v) : w_f \in [\alpha, 1/3]\}$.
- Tiny $T := \{f \in \delta(v) : w_f \in (1/10, \alpha)\}$.

First we have the following observation.

▶ **Observation 15.** *In any bin packing solution, in any bin there can be at most one item from L, two items from $L \cup M$, three items from $L \cup M \cup S$ and nine items from $L \cup M \cup S \cup T$.*

Now let us claim the following.

▶ **Claim 16.** *For any tight color i at v, all edges added in Step (4) of the algorithm are in S. As a corollary, there is at most one edge incident on v with color i that is in $L \cup M \cup T$ and it can only be added in Step (2) of the algorithm.*

Proof. From Claim 9, it follows that all edges in $L \cup M$ must be included in F in Step (2) of the algorithm. On the other hand, all edges colored in Step (4) have weight more than α, they can not be in T. Hence, only edges in S are colored in Step (4). Edges in $L \cup M \cup T$ are colored in Step (3). In Step (3) of the algorithm, we include at most one edge of F incident at any vertex in each F_i. Thus each color class obtains at most one edge incident at each vertex from F and therefore, from $L \cup M \cup T$. ◀

Using the observation and the claim, we itemize the configuration of each of the tight colors depending on the size of edges with that color. Note that in this case tight colors have weight more than $1 - \alpha \geq 1 - 1/3 = 2/3$.

1. If the tight color has a single edge f. Then we have that $w_f > 2/3$ and only possibility is $i)(L)$; Here by (L), we again denote that the bin contains only one item and that item is an item from the set L.
2. If the tight color contains exactly two edges. From Claim 16, the bin can contain at most one item from $L \cup M \cup T$. On the other hand, (S, S) or (T, S) has weight $\leq 2/3$. So the bin can contain at most one item from S and one item from $L \cup M$. Thus the possible size types of these edges are $ii)(L, S); iii)(M, S)$; As above, by (L, S) we denote that the bin contains only two items: exactly one item from set L and exactly one item from S.
3. If the tight color contains three edges. From Claim 16, the bin can contain at most one item from $L \cup M \cup T$. However if the bin contains an item from L, the sum of weights of an item from L and two items from S exceeds one. Thus the possible size types of these edges are $iv)(M, S, S); v)(S, S, S); vi)(T, S, S)$.

Now consider the following configuration LP based on the items at v: $LP_{bin}(v)$:

$$min \quad \sum_{i=1}^{19} y_i$$

$$\begin{aligned}
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 &\geq \beta m && (13)\\
y_{14} &\geq x_1 && (14)\\
y_8 + y_9 + y_{10} + y_{15} &\geq x_2 && (15)\\
y_3 + y_7 + y_9 + y_{11} + 2y_{12} + y_{16} &\geq x_3 && (16)\\
y_2 + 2y_4 + y_6 + y_{10} + y_{11} + 2y_{13} + y_{17} &\geq x_4 && (17)\\
3y_1 + 2y_2 + 2y_3 + y_4 + 2y_5 + y_6 + y_7 + y_8 + y_{18} &\geq x_2 + x_3 + 2x_4 + 3x_5 + 2x_6 && (18)\\
(3y_5 + 3y_6 + 3y_7 + y_8 + y_9 + y_{10} + 3y_{11} + 3y_{12} & \\
+3y_{13} + 3y_{14} + 4y_{15} + 6y_{16} + 6y_{17} + 6y_{18} + 9y_{19}) &\geq x_6 && (19)\\
y_j, x_k &\geq 0 \quad \forall j \in [19], k \in [6] && (20)
\end{aligned}$$

▶ **Lemma 17.** *The optimal number of unit sized bins needed to pack all edges incident at u is at least the optimum value of $LP_{bin}(v)$.*

Proof. Given a feasible packing of edges incident at v in at most m unit sized bins, we construct a feasible solution $(\bar{x}, \bar{y})$ to the linear programming relaxation whose objective is at most the number of unit sized bins needed in the packing. In the feasible solution $(\bar{x}, \bar{y})$, the variables $\bar{x}$ are constructed using the coloring given by the algorithm. The variables $\bar{y}$ are constructed using the optimal bin packing.

We first define the variables $\bar{x}$. Let $\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, \bar{x}_6$ be the number of tight colors at v of type $(L), (L,S), (M,S), (M,S,S), (S,S,S), (T,S,S)$, respectively. Since the coloring of the edges incident at v is one of the six types described above and there are at least βm tight colors at v, we have that $\sum_{i=1}^{6} \bar{x}_i \geq \beta m$ and thus the solution satisfies constraint (13).

To construct the solution $\bar{y}$, we will group the bins in the optimal bin packing solutions depending on the subset of items present in them into nineteen classes and the number of bins in each class will define the variables $\bar{y}$. The constraints (14)-(19) will correspond to making sure that the optimal bin packing solution has appropriate number of items of each size type.

To define the 19 different classes of bin types in the optimal solution, we need to further classify items according to size. Let $L_1, L_2 \subseteq L$ be the set of large edges that appear in the configurations of the type (L) and (L,S), respectively, in the tight colors.

As for any item $l_1 \in L_1$, $w_{l_1} + \alpha > 1$. We get,

$$w_{l_1} > 1 - \alpha \geq 1 - 1/3 = 2/3. \tag{21}$$

Let M_1, M_2 be the set of medium edges that appear in the tight colors of type (M,S) and (M,S,S), respectively.

We now have the following claim where we characterize the possible bin configurations. We show that each bin contains items which correspond to one of 19 possible configurations or their subsets.

▶ **Claim 18.** *Consider any feasible bin packing of edges incident at v restricted to edges in $L \cup M \cup S \cup T$. Then each bin must contain items which correspond to a subset of one of the following 19 configurations.*

$C_1 : (S,S,S)$	$C_2 : (M_2,S,S)$	$C_3 : (M_1,S,S)$
$C_4 : (M_2,M_2,S)$	$C_5 : (S,S,T,T,T)$	$C_6 : (M_2,S,T,T,T)$
$C_7 : (M_1,S,T,T,T)$	$C_8 : (L_2,S,T)$	$C_9 : (L_2,M_1,T)$
$C_{10} : (L_2,M_2,T)$	$C_{11} : (M_1,M_2,T,T,T)$	$C_{12} : (M_1,M_1,T,T,T)$
$C_{13} : (M_2,M_2,T,T,T)$	$C_{14} : (L_1,T,T,T)$	$C_{15} : (L_2,T,T,T,T)$
$C_{16} : (M_1,T,T,T,T,T,T)$	$C_{17} : (M_2,T,T,T,T,T,T)$	$C_{18} : (S,T,T,T,T,T,T)$
$C_{19} : (T,T,T,T,T,T,T,T,T)$		

Proof. Observe that since item in L have weight more than $\frac{1}{2}$, items in M have weight more than $\frac{1}{3}$, items in S have weight more than $\frac{1}{4}$ and items in T have weight more than $\frac{1}{10}$, there can be at most one item from L, two items items from $L \cup M$, at most three items in total from $L \cup M \cup S$ and at most nine items from $L \cup M \cup S \cup T$ in any feasible packing.

1. Bins with three items from $D := L \cup M \cup S$. As the sum of weights of three elements from D is more than $3\alpha > 0.9$, elements from T can not appear in these bins as $3\alpha + w_f > 1$ for any $f \in T$. Moreover, configurations which contain at least one item of L cannot have three items from D without the weight exceeding one. Thus, the packing can contain only items from M and S. When the bin contains only S items, it corresponds to configuration C_1. Packings which contain one item from $M_1 \cup M_2$ and two items from S are exactly the configurations C_2, C_3.

Now let us consider the case when we have two items from $M_1 \cup M_2$. First observe that for each $h \in M_1$, there exists a $s \in S$ such that (h, s) are the only edges colored with a tight color. Thus we have that $w_h + w_s > 1 - \alpha$. But then for any other $h' \in M_1 \cup M_2$ and $s' \in S$, we have that

$$w_h + w_{h'} + w_{s'} \geq w_h + w_s + \alpha > 1 \tag{22}$$

where the inequality follows since $w_{h'} \geq w_s$ and $w_{s'} \geq \alpha$. This implies that configurations of type (M_1, M_1, S) or (M_1, M_2, S) are not feasible. Hence, the only possible remaining configuration is C_4.

2. Bins with two items from D. Here we consider maximal configurations which are not subsets of configurations which contain three items from D. When the configuration contains two items $s_1, s_2 \in S$, we have that $w_{s_1} + w_{s_2} > 0.6$ and thus the only maximal configuration is C_5. Configurations C_6, C_7 cover the case when the configuration contains one item from S and one item from M. Let $l \in L_1$. Since l appears alone in a tight color, we have that $w_l + \alpha > 1$. Since every item in $M \cup S$ has weight at least α, there is no valid configuration with two items from D such that one of them is in L_1. If the configuration contains $l_2 \in L_2$ and $g \in M_1 \cup M_2 \cup S$, it can at most contain one element $t \in T$ as $w_{l_2} + w_g + w_t > 0.9$. Thus configurations C_8 cover the case when there is one item from S and one item from L. Now we are left with cases when there are no S items in the bin. If there is one L item and one M item, C_9, C_{10} cover such possibilities.
   Similarly if the configuration contains two items from M, it can contain at most 3 elements from T. Configurations C_{11}, C_{12}, C_{13} cover all such the possibilities.
3. Bins with one item from D. Here we consider configurations which are not subsets of configurations which contain at least two items from D. Note that as for any item $l_1 \in L_1$, from inequality (21), $w_{l_1} > 2/3$. Thus (L_1, T, T, T) is the maximal configuration containing one L_1 item. C_{14} is the corresponding configuration. The other four possible configurations are $C_{15}, C_{16}, C_{17}, C_{18}$ where the bins contain one item from L_2, M_1, M_2, S respectively. In these case the number of T items are upper bounded by $4, 6, 6, 6$ respectively from the lower bound of size of items in the corresponding classes in D.
4. Bins with no item from D. Only possible maximal configuration is C_{19}. ◀

We map each configuration in the optimal bin packing solution to one of types C_i where the configuration is either C_i or its subset. Let $\bar{y}_i$ denote the number of bins mapped to type C_i for each $1 \leq i \leq 19$. We now count the number of items of each type to show feasibility of the constraints of the linear program.

Constraint (14). Items of type L_1 equal $\bar{x}_1$ and can only be contained in configuration C_{14}. Thus we have $\bar{y}_{14} \geq \bar{x}_1$.

Constraint (15). Similarly, items of type L_2 equal $\bar{x}_2$. They are contained in configurations C_8, C_9, C_{10} and C_{15}. Thus we have $\bar{y}_8 + \bar{y}_9 + \bar{y}_{10} + \bar{y}_{15} \geq \bar{x}_2$.

Constraint (16). Items of type M_1 equal $\bar{x}_3$ and are contained once in configurations $C_3, C_7, C_9, C_{11}, C_{16}$ and twice in configuration C_{12}. Thus we have $\bar{y}_3 + \bar{y}_7 + \bar{y}_9 + \bar{y}_{11} + 2\bar{y}_{12} + \bar{y}_{16} \geq \bar{x}_3$.

Constraint (17). Items of type M_2 equal $\bar{x}_4$ and are contained once in configurations $C_2, C_6, C_{10}, C_{11}, C_{17}$ and twice in configurations C_4, C_{13}. Thus we have $\bar{y}_2 + 2\bar{y}_4 + \bar{y}_6 + \bar{y}_{10} + \bar{y}_{11} + 2\bar{y}_{13} + \bar{y}_{17} \geq \bar{x}_4$ satisfying constraint (17).

Constraint (18). Items of type S equal $\bar{x}_2 + \bar{x}_3 + 2\bar{x}_4 + 3\bar{x}_5 + 2\bar{x}_6$ and occur once in configurations $C_4, C_6, C_7, C_8, C_{18}$, twice in configurations C_2, C_3, C_5 and thrice in C_1. Thus, we have $3\bar{y}_1 + 2\bar{y}_2 + 2\bar{y}_3 + \bar{y}_4 + 2\bar{y}_5 + \bar{y}_6 + \bar{y}_7 + \bar{y}_8 + \bar{y}_{18} \geq \bar{x}_2 + \bar{x}_3 + 2\bar{x}_4 + 3\bar{x}_5 + 2\bar{x}_6$.

Constraint (19). Items of type T equal $\bar{x}_6$ and occur once in configurations C_8, C_9, C_{10}, thrice in configurations $C_5, C_6, C_7, C_{11}, C_{12}, C_{13}, C_{14}$, four times in configuration C_{15}, six times in configurations C_{16}, C_{17}, C_{18} and nine times in configurations C_{19}. Thus, we have $(3\bar{y}_5+3\bar{y}_6+3\bar{y}_7+\bar{y}_8+\bar{y}_9+\bar{y}_{10}+3\bar{y}_{11}+3\bar{y}_{12}+3\bar{y}_{13}+3\bar{y}_{14}+4\bar{y}_{15}+6\bar{y}_{16}+6\bar{y}_{17}+6\bar{y}_{18}+9\bar{y}_{19}) \geq \bar{x}_6$. This implies that $(\bar{x}, \bar{y})$ is a feasible solution to $LP_{bin}(v)$ and its objective equals the number of bins needed to pack the edges incident at v in unit sized bins. Thus we have the lemma. ◀

We now show a contradiction by showing the optimal value of the $LP_{bin}(v)$ is more than m.

▶ **Lemma 19.** *The optimal solution to the $LP_{bin}(v)$ is strictly more than m.*

Proof. We prove this by considering the dual linear program of the $LP_{bin}(v)$. Since every feasible solution to the dual LP gives a lower bound on the objective of the primal $LP_{bin}(v)$, it is enough to exhibit a feasible dual solution of objective strictly more than m to prove the lemma. Now the dual of the $LP_{bin}(v)$ is given below:

$$
\begin{array}{ll}
max \quad \beta m \cdot v_1 & \\
\text{Subject to:} & \\
v_1 - v_2 \leq 0, & v_1 - v_3 - v_6 \leq 0, \\
v_1 - v_4 - v_6 \leq 0, & v_1 - v_5 - 2v_6 \leq 0, \\
v_1 - 3v_6 \leq 0, & v_1 - 2v_6 - v_7 \leq 0, \\
3v_6 \leq 1, & v_5 + 2v_6 \leq 1, \\
v_4 + 2v_6 \leq 1, & 2v_5 + v_6 \leq 1, \\
2v_6 + 3v_7 \leq 1, & v_5 + v_6 + 3v_7 \leq 1, \\
v_4 + v_6 + 3v_7 \leq 1, & v_3 + v_6 + v_7 \leq 1, \\
v_3 + v_4 + v_7 \leq 1, & v_3 + v_5 + v_7 \leq 1, \\
v_4 + v_5 + 3v_7 \leq 1, & 2v_4 + 3v_7 \leq 1, \\
2v_5 + 3v_7 \leq 1, & v_2 + 3v_7 \leq 1, \\
v_3 + 4v_7 \leq 1, & v_4 + 6v_7 \leq 1, \\
v_5 + 6v_7 \leq 1, & v_6 + 6v_7 \leq 1, \\
9v_7 \leq 1, & v_i \geq 0 \quad \forall i \in [7].
\end{array}
$$

A feasible dual solution is: $v_1 = \frac{9}{13}, v_2 = \frac{9}{13}, v_3 = \frac{7}{13}, v_4 = \frac{5}{13}, v_5 = \frac{1}{13}, v_6 = \frac{4}{13}, v_7 = \frac{1}{13}$. Thus dual optima is at least $\beta m \cdot \frac{9}{13} > m$. Thus, we need more than m bins to pack all items incident at v, a contradiction. ◀

This completes the proof of Lemma 14. ◀

Therefore, the proof of Theorem 2 is complete. ◀

If we assume that all edges have weight more than $1/4$, then similar analysis will attain $2.2m$ colors are sufficient. For the proof, we refer the readers to [13].

▶ **Theorem 20.** *If all edges have weight more than $1/4$, then there is a polynomial time algorithm for the* WEIGHTED BIPARTITE EDGE COLORING *problem which returns a proper weighted coloring using at most $\lceil 2.2m \rceil$ colors where m is denotes the maximum over all the vertices of the number of unit-sized bins needed to pack the weights of incident edges, i.e., $\mu_{(\frac{1}{4},1]}(m, r) \leq \lceil 2.2m \rceil$.*

3 Conclusion

Considering the case $1/4 \geq \alpha > 1/5$ separately, might improve the bound by more case analysis. However we can attain at most $35m/16 \approx 2.19m$ by that. Finding a better approximation algorithm (independent of m) or inapproximability, and extending our techniques to general graphs will be interesting.

Acknowledgements. We thank Prasad Tetali for helpful discussions.

References

1 John Beetem, Monty Denneau, and Don Weingarten. The gf11 supercomputer. In *International Symposium on Computer architecture*, pages 108–115, 1985.

2 Shun-Ping Chung and Keith W. Ross. On nonblocking multirate interconnection networks. *SIAM Journal on Computing*, 20(4):726–736, 1991.

3 C. Clos. A study of nonblocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

4 Edward G. Coffman Jr, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

5 J. Correa and M. X. Goemans. Improved bounds on nonblocking 3-stage clos networks. *SIAM Journal of Computing*, 37:870–894, 2007.

6 D. Z. Du, B. Gao, F. K. Hwang, and J. H. Kim. On multirate rearrangeable clos networks. *SIAM Journal of Computing*, 28(2):463–470, 1999.

7 Uriel Feige and Mohit Singh. Edge coloring and decompositions of weighted graphs. In *ESA*, pages 405–416, 2008.

8 Michael R. Garey, Ronald L. Graham, and Jeffrey D. Ullman. Worst-case analysis of memory allocation algorithms. In *STOC*, pages 143–150. ACM, 1972.

9 Rebecca Hoberg and Thomas Rothvoß. A logarithmic additive integrality gap for bin packing. *CoRR*, abs/1503.08796, 2015.

10 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

11 A. Itoh, W. Takahashi, H. Nagano, M. Kurisaka, and S. Iwasaki. Practical implementation and packaging technologies for a large-scale atm switching system. *Journal of Selected Areas in Communications*, 9:1280–1288, 1991.

12 Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320. IEEE, 1982.

13 Arindam Khan. *Approximation Algorithms for Multidimensional Bin Packing*. PhD thesis, Georgia Institute of Technology, Atlanta, 2015.

14 D. König. Graphok és alkalmazásuk a determinánsok és a halmazok elméletére. *Mathematikai és Természettudományi Ertesito*, 34:104–119, 1916.

15 Riccardo Melen and Jonathan S. Turner. Nonblocking multirate distribution networks. *IEEE Transactions on Communications*, 41(2):362–369, 1993.

16 Hung Q. Ngo and Van H. Vu. Multirate rearrangeable clos networks and a generalized edge-coloring problem on bipartite graphs. *SIAM J. Comput.*, 32(4):1040–1049, 2003.

17 Thomas Rothvoß. Approximating bin packing within o(log opt * log log opt) bins. In *FOCS*, pages 20–29, 2013.

18 Claude E. Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(2):148–151, 1949.

19 T. D. Slepian. Two theorems on a particular crossbar switching. *unpublished manuscript*, 1958.

20 Michael Stiebitz, Diego Scheide, Bjarne Toft, and Lene M. Favrholdt. *Graph Edge Coloring: Vizing's Theorem and Goldberg's Conjecture*, volume 75. John Wiley & Sons, 2012.

21 Peter Guthrie Tait. Remarks on the colouring of maps. *Proc. Roy. Soc. Edinburgh*, 10(729):501–503, 1880.

22 V. G. Vizing. On an estimate of the chromatic class of a p-graph (in russian). *Diskret. Analiz*, 3:23–30, 1964.

Deciding Orthogonality in Construction-A Lattices

Karthekeyan Chandrasekaran [1], Venkata Gandikota[2], and Elena Grigorescu[3]

1 **University of Illinois, Urbana-Champaign, IL, USA**
karthe@illinois.edu
2 **Purdue University, West Lafayette, IN, USA**
vgandiko@purdue.edu
3 **Purdue University, West Lafayette, IN, USA**
elena-g@purdue.edu*

Abstract

Lattices are discrete mathematical objects with widespread applications to integer programs as well as modern cryptography. A fundamental problem in both domains is the Closest Vector Problem (popularly known as CVP). It is well-known that CVP can be easily solved in lattices that have an orthogonal basis *if* the orthogonal basis is specified. This motivates the orthogonality decision problem: verify whether a given lattice has an orthogonal basis. Surprisingly, the orthogonality decision problem is not known to be either NP-complete or in P.

In this paper, we focus on the orthogonality decision problem for a well-known family of lattices, namely Construction-A lattices. These are lattices of the form $C + q\mathbb{Z}^n$, where C is an error-correcting q-ary code, and are studied in communication settings. We provide a complete characterization of lattices obtained from binary and ternary codes using Construction-A that have an orthogonal basis. This characterization leads to an efficient algorithm solving the orthogonality decision problem, which also finds an orthogonal basis if one exists for this family of lattices. We believe that these results could provide a better understanding of the complexity of the orthogonality decision problem in general.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Orthogonal Lattices, Construction-A, Orthogonal Decomposition, Lattice isomorphism

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.151

1 Introduction

A lattice is the set of integer linear combinations of a set of basis vectors $B \in \mathbb{R}^{m \times n}$, namely $L = L(B) = \{xB \mid x \in \mathbb{Z}^m\}$. Lattices are well-studied fundamental mathematical objects that have been used to model diverse discrete structures such as in the area of integer programming [7], or in factoring integers [14] and factoring rational polynomials [8]. In a groundbreaking result, Ajtai [1] demonstrated the potential of computational problems on lattices to cryptography, by showing average case/worst case equivalence between lattice problems related to finding short vectors in a lattice. This led to renewed interest in the complexity of two fundamental lattice problems: the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). Concretely, in SVP, given a basis B one is asked to output a shortest non-zero vector in the lattice, and in CVP, given a basis B and a target $t \in \mathbb{R}^n$, one is asked to output a lattice vector closest to t.

* The research of V. G. and of E. G. was partially funded by Purdue Research Foundation grants.

Both SVP and CVP are NP-hard even to approximate up to subpolynomial factors (see [12] for a survey), and a great deal of research in complexity theory has been devoted to finding families of lattices for which SVP/CVP are easy. A simplest lattice for which CVP is easy is $\mathbb{Z}^n$: indeed, finding the closest lattice vector to a target $t \in \mathbb{R}^n$ amounts to rounding the entries of t to the nearest integer. Surprisingly, given an arbitrary basis B, it is not known how to efficiently verify whether the lattice generated by B is isomorphic to $\mathbb{Z}^n$ upto an orthogonal transformation. Further, given an arbitrary basis for a lattice, it is not known how to decide efficiently if the lattice has an orthogonal basis (an orthogonal basis is a basis in which all vectors are pairwise orthogonal). Similar to the case of $\mathbb{Z}^n$, having access to an orthogonal basis leads to an efficient algorithm to solve CVP, but finding an orthogonal basis given an arbitrary basis appears to be non-trivial, with no known efficient algorithms.

Deciding if a lattice is equivalent to $\mathbb{Z}^n$, and deciding if a lattice has an orthogonal basis, are special cases of the more general Lattice Isomorphism Problem (LIP). In LIP, given lattices L_1 and L_2 presented by their bases, one is asked to decide if they are isomorphic, meaning if there exists an orthogonal transformation that takes one to the other. LIP has been studied in [13, 15, 6] and is known to have a $n^{O(n)}$ algorithm [6]. Recent results of [10, 9] show that in certain highly symmetric lattices, isomorphism to $\mathbb{Z}^n$ can be decided efficiently.

The complexity of LIP is not well understood, and is part of the broader study of isomorphism between mathematical objects, of which Graph Isomorphism (GI) is a well-known elusive problem [2]. Interestingly, there is a polynomial time reduction from GI to LIP [15].

Given that LIP, deciding isomorphism to $\mathbb{Z}^n$, and deciding whether a lattice has an orthogonal basis appear to be difficult problems for arbitrary input lattices, it is natural to address families of lattices where these problems are solvable efficiently. In this work, we focus on the problem of deciding orthogonality for a particular family of lattices, commonly known as Construction-A lattices [5]. A Construction-A lattice L is obtained from a linear error-correcting code C over a finite field of q elements[1] (denoted $\mathbb{F}_q$) as $L = C + q\mathbb{Z}^n$. We resolve the problem of deciding orthogonality in Construction-A lattices for $q = 2$ and $q = 3$ showing an efficient algorithm. In addition, the algorithm outputs an orthogonal basis of the lattice if such a basis exists.

Our main technical contribution is a decomposition theorem for Construction-A lattices that admit an orthogonal basis. A natural way to obtain an orthogonal Construction-A lattice is by taking direct products of lower dimensional orthogonal lattices. We show that this is the only possible way and that the lower dimensional orthogonal lattices indeed have constant dimension. We believe that our contributions are a step towards gaining a better understanding of lattice isomorphism problems for more general classes of lattices.

Extending our results to values $q > 3$ might require new techniques. For higher q, a decomposition characterization seems to require a complete characterization of *weighing matrices* of weight q which is a known open problem. In particular, a direct product decomposition characterization of weighing matrices for the case of $q = 4$ is known. However, the parts in the direct product decomposition may not be of constant dimension. As a consequence, the lattice decomposition theorem, if true, would only suggest that orthogonal Construction-A lattices necessarily decompose into direct products of lattices, which could be high-dimensional. So designing an efficient algorithm for the orthogonality decision problem exploiting the direct product decomposition characterization appears to be non-trivial.

[1] The term 'Construction-A' strictly refers to the case $q = 2$, but we will not make the distinction in this paper.

1.1 Our results and techniques

As mentioned above, we start by showing a structural decomposition of orthogonal lattices of the form $C + 2\mathbb{Z}^n$ and $C + 3\mathbb{Z}^n$ into constant-size orthogonal lattices. We remark that the decomposition holds up to permutations of the coordinates, and we use the notation $C_1 \cong C_2$ and $L_1 \cong L_2$ to denote the equivalence of codes and lattices under permutation of coordinates. We use the notation $L_1 \otimes L_2$ to denote the direct product of two lattices.

▶ **Theorem 1.** *Let $L_C = C + 2\mathbb{Z}^n$ be a lattice obtained from a binary linear code $C \subseteq \mathbb{F}_2^n$. Then the following statements are equivalent:*

1. *L_C is orthogonal.*
2. *$L_C \cong \otimes_i L_i$, where each L_i is either $\mathbb{Z}$, or $2\mathbb{Z}$, or the 2-dimensional lattice generated by the rows of the matrix $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.*
3. *$C \cong \otimes_i C_i$, where each C_i is either a length-1 binary linear code $\subseteq \{0, 1\}$, or the length-2 binary linear code $\{00, 11\}$.*

The decomposition characterization leads to an efficient algorithm to verify if a given lattice obtained from a binary linear code using Construction-A is orthogonal. For the purposes of this algorithmic problem, the input consists of a basis to the lattice. The algorithm finds the component codes given by the characterization thereby computing the orthogonal basis for such a lattice.

▶ **Theorem 2.** *Given a basis for a lattice L obtained from a binary linear code $C \subseteq \mathbb{F}_2^n$ using Construction-A, there exists an algorithm running in time $O(n^6)$ that verifies if L is orthogonal, and if so, it outputs an orthogonal basis.*

We obtain a similar decomposition and algorithm for lattices obtained from ternary codes. For succinctness of presentation we define the following integer matrix:

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 0 & -1 & -1 \\ 0 & 1 & -1 & 1 \end{bmatrix}.$$

▶ **Theorem 3.** *Let $L_C = C + 3\mathbb{Z}^n$ be a lattice obtained from a ternary linear code $C \subseteq \mathbb{F}_3^n$. Then the following statements are equivalent:*

1. *L_C is orthogonal.*
2. *$L_C \cong \otimes_i L_i$, where each L_i is either $\mathbb{Z}$, or $3\mathbb{Z}$, or the 4-dimensional lattice generated by the rows of a matrix $\mathcal{T}(M)$ obtained from M by negating some subset of columns.*
3. *$C \cong \otimes_i C_i$, where each C_i is either a linear length-1 ternary code, or the linear length-4 ternary code generated by the rows of $(\mathcal{T}(M) \bmod 3) \in \mathbb{F}_3^{4\times 4}$, where $\mathcal{T}(M)$ is obtained from M by negating some subset of its columns.*

▶ **Theorem 4.** *Given a basis for a lattice L obtained from a ternary linear code using Construction-A, there exists an algorithm running in time $O(n^8)$ that verifies if L is orthogonal, and if so, it outputs an orthogonal basis.*

In the interests of space, we prove Theorems 3 and 4 here and defer the proofs of Theorems 1 and 2 to the full version of this work [4].

2 Preliminaries

We denote by $[n]$ the set of positive integers up to n, the $n \times n$ identity matrix by I_n and its j^{th} row by e_j. For a vector $b \in \mathbb{R}^n$, let b_j denote its j^{th} coordinate, and $\|b\|$ denote its ℓ_2 norm.

A lattice $L \subseteq \mathbb{R}^n$ is said to be of full rank if it is generated by n linearly independent vectors. A lattice L is said to be orthogonal if it has a basis B such that the rows of B are pairwise orthogonal vectors. A lattice L is *integral* if it is contained in $\mathbb{Z}^n$, namely any basis for L only consists of integer vectors.

We will denote by $\mathbb{F}_q$ a finite field with q elements. A *linear* code C of length n over $\mathbb{F}_q$ is a vectorspace $C \subseteq \mathbb{F}_q^n$. A linear code is specified by a generator matrix G that consists of linearly independent vectors in $\mathbb{F}_q^n$. If $C \subseteq \mathbb{F}_2^n$ it is called a *binary* code, and if $C \subseteq \mathbb{F}_3^n$ it is called a *ternary* code.

The Construction-A of a lattice L_C from a linear code $C \subseteq \mathbb{F}_q^n$, where q is a prime, is defined as $L_C := \{c + q \cdot z \mid c \in \phi(C), z \in \mathbb{Z}^n\}$, where ϕ is the the (real embedding) mapping $i \in \mathbb{F}_q \mapsto i \in \mathbb{Z}$. Construction-A is often abbreviated as $L_C = C + q\mathbb{Z}^n$.

For any vector $v = (v_1, \cdots, v_n) \in \mathbb{Z}^n$ define $v \bmod q = (v_1 \bmod q, \cdots, v_n \bmod q) \in \mathbb{F}_q^n$.

▶ **Claim 5.** *Let q be a prime. If $q\mathbb{Z}^n \subseteq L$ then $C = L \bmod q$ is a linear code over $\mathbb{F}_q$.*

Proof. Let $v \in L$ and $v = (v \bmod q) + qz$ for some $z \in \mathbb{Z}^n$, where here we abuse notation and view $v \bmod q$ as embedded into the integers, instead of a vector in $\mathbb{F}_q^n$. Since $q\mathbb{Z}^n \subseteq L$, it follows that $v - qz = v \bmod q \in L$. To show that $C = L \bmod q$ is a linear code over $\mathbb{F}_q$, let $c_1, c_2 \in C$. Then $c_1 + c_2 \in L$ (where the addition is over $\mathbb{Z}$), and so $(c_1 + c_2) \bmod q \in C$. ◀

We will use the following immediate claim about product of lattices generated from codes.

▶ **Claim 6.** *Let $L = C + q\mathbb{Z}^n$, for some q-ary linear code $C \subseteq \mathbb{F}_q^n$. If $L \cong L_1 \otimes L_2$, and $L_1 \subseteq \mathbb{Z}^k$, then $L_1 \cong C_1 + q\mathbb{Z}^k$ and $L_2 \cong C_2 + q\mathbb{Z}^{n-k}$, for q-ary linear codes C_1 and C_2 that are projections of C on the coordinates corresponding to L_1 and L_2 respectively.*

A matrix U is *unimodular* if $U \in \mathbb{Z}^{n \times n}$ and $\det(U) \in \{\pm 1\}$. Two different bases B_1, B_2 give rise to the same lattice if and only if there exists a unimodular matrix U such that $B_1 = UB_2$.

The *Hermite Normal Form (HNF) basis* for a full rank lattice $L \subseteq \mathbb{R}^n$ is a square, non-singular, upper triangular matrix $B \subseteq \mathbb{R}^{n \times n}$ such that off-diagonal elements satisfy : $0 \leq b_{i,j} < b_{j,j}$ for all $1 \leq i < j \leq n$.

▶ **Fact 7.** *[11] There exists an efficient algorithm which on input a set of rational vectors B, computes a basis for the lattice generated by B: the algorithm simply computes the unique HNF basis of the lattice generated by B.*

We note that $L_C = C + q\mathbb{Z}^n$ contains $q\mathbb{Z}^n$ as a sublattice and hence it is a full rank lattice.

▶ **Fact 8.** *A basis B for the lattice L_C specified by the generator matrix G for the code C can be computed efficiently by taking the HNF of the matrix $\begin{bmatrix} G \\ qI_n \end{bmatrix}$. Conversely, given a basis B of L_C, the generator matrix for C can be computed efficiently by finding a basis for $B \bmod q$ by row reduction over $\mathbb{F}_q$.*

A *weighing matrix* of order n and weight k is a $n \times n$ matrix with entries in $\{0, 1, -1\}$ such that each row and column has exactly k non-zero entries and the row vectors are orthogonal to each other. By definition, a weighing matrix W satisfies $WW^T = kI_n$. For matrices $A \in \mathbb{R}^{n_1 \times n_1}$ and $B \in \mathbb{R}^{n_2 \times n_2}$, we denote the $(n_1 + n_2) \times (n_1 + n_2)$-dimensional block-diagonal matrix obtained using blocks A and B by $A \otimes B$. We will use the following characterization of weighing matrices of weight 2 and 3. Please refer to the full version [4] for the proofs of Theorem 9 and Theorem 10.

▶ **Theorem 9** ([3]). *A matrix W is a weighing matrix of order n and weight 2 if and only if W can be obtained from*

$$\otimes_{i=1}^{n/2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

by negating some rows and columns and by interchanging some rows and columns.

▶ **Theorem 10** ([3]). *A matrix W is a weighing matrix of order n and weight 3 if and only if W can be obtained from $\otimes_{i=1}^{n/4} M$ by negating some rows and columns and by interchanging some rows and columns.*

3 Orthogonal Lattices from Ternary Codes

In this section we focus on lattices obtained from ternary linear codes using Construction-A. In Section 3.1, we show that any orthogonal lattice obtained from a ternary linear codes by Construction-A is equivalent to a product lattice whose components are one-dimensional or four-dimensional. In Section 3.2, we show that given a lattice obtained from a ternary linear code by Construction-A, there exists an efficient algorithm to verify if the lattice is orthogonal.

3.1 Decomposition Characterization

We prove Theorem 3 in this subsection.

Proof of Theorem 3. We show that (1) $\equiv$ (2) and (2) $\equiv$ (3) to complete the equivalence of the three statements.

(1) $\equiv$ (2): We show that $L_C = C + 3\mathbb{Z}^n$ is orthogonal if and only if it decomposes into direct product of lower dimensional orthogonal lattices, $L_C \cong \otimes_i L_i$.
If $L_C \cong \otimes_i L_i$ such that each L_i is orthogonal, then L_C is also orthogonal, since L_C has a block diagonal basis where each block is itself an orthogonal matrix (by definition, a 1×1-dimensional matrix is orthogonal) .

We prove the other side by induction on the dimension, n of the lattice L_C. For the base case consider $n = 1$. Since L is integral, contains $3\mathbb{Z}$ and is of the form $C + 3\mathbb{Z}$ for some ternary code C, it follows that L has to be either $\mathbb{Z}$ or $3\mathbb{Z}$. Let us assume the induction hypothesis for all $n - 1$ or lower dimensional orthogonal lattices obtained from ternary linear codes using construction-A.

Let L_C be an n-dimensional orthogonal lattice and B be its orthogonal basis. Since L_C is an integral lattice, B has only integral entries. The next two claims summarize certain properties of the entries of the basis matrix B.

▶ **Claim 11.** *For every row b of B and for every $j \in [n]$, we have that $3|b_j| \in \{0, \|b\|^2, 3\|b\|^2\}$.*

Proof. Since B is an orthogonal basis, $BB^T = D$, where D is the diagonal matrix with $d_i = \|b^{(i)}\|^2$, where $b^{(i)}$ denotes the i^{th} basis vector.

$$D = \begin{bmatrix} \|b^{(1)}\|^2 & 0 & 0 & \cdots & 0 \\ 0 & \|b^{(2)}\|^2 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & \|b^{(n)}\|^2 \end{bmatrix}$$

We know that $3\mathbb{Z}^n \subseteq L_C$ so, $3e_j \in L_C$ for every $j \in [n]$. Therefore, there is an integral matrix $X \in \mathbb{Z}^{n\times n}$ such that $XB = 3I_n$, i.e. $3B^{-1} \in \mathbb{Z}^{n\times n}$. Since we started with an orthogonal basis B,

$$B^{-1} = B^T D^{-1} \in \frac{1}{3}\mathbb{Z}^{n\times n}.$$

Each column of $B^T D^{-1}$ is given by $b/\|b\|^2$, where b is a basis vector. Therefore, for any $j \in [n], 3b_j$ is a multiple of $\|b\|^2$, formally

$$3b_j \equiv 0 \bmod \|b\|^2 \text{ for all } j \in [n], \text{ and rows } b \text{ of } B. \tag{1}$$

Since b_j is integral and $|b_j| \leq \|b\|^2$ for every $j \in [n]$, it follows from the above equation that $3|b_j| \in \{0, \|b\|^2, 2\|b\|^2, 3\|b\|^2\}$. Suppose there exists $j \in [n]$ such that $3|b_j| = 2\|b\|^2$. Since b is a basis vector, it follows that b is not all zeroes. Hence $b_j \neq 0$. We can re-write the condition $3|b_j| = 2\|b\|^2$ as $3|b_j| = 2\sum_{i=1}^n b_i^2$. Rearranging the terms, we have

$$|b_j|\ (3 - 2|b_j|) = 2\sum_{i\neq j} b_i^2.$$

Since the RHS is a sum of squares, it is always non-negative. The LHS is non-zero since $b_j \in \mathbb{Z} \setminus \{0\}$. So the LHS should be strictly positive. Therefore, $|b_j| \in (0, 3/2) \cap \mathbb{Z}$ and hence $|b_j| = 1$. However, this implies that $\sum_{i\neq j} b_i^2 = 1/2$, contradicting the fact that b is integral. Hence, $3||b_j|| = 2\|b\|^2$ is impossible. ◀

▶ **Claim 12.** *Let b be a row of B.*

1. *If there exists $j \in [n]$ such that $3|b_j| = 3\|b\|^2$, then $b_j = \pm 1$ and $b_{j'} = 0$ for every $j' \in [n] \setminus \{j\}$.*
2. *If there exists $j \in [n]$ such that $3|b_j| = \|b\|^2$ and $b_j = \pm 3$, then $b_{j'} = 0$ for every $j' \in [n] \setminus \{j\}$.*
3. *If there exists $j \in [n]$ such that $3|b_j| = \|b\|^2$ and $b_j = \pm 1$, then there exist $j_1, j_2 \in [n] \setminus \{j\}$, such that $|b_{j_1}| = |b_{j_2}| = 1$ and $b_{j'} = 0$ for every $j' \in [n] \setminus \{j, j_1, j_2\}$.*
4. *If there exists $j \in [n]$ such that $3|b_j| = \|b\|^2$, then $b'_j \in \{0, \pm 1, \pm 3\}$ for every $j' \in [n]$.*

Proof.

1. Since, $\|b\|^2 = \sum_{i=1}^n b_i^2$, and each $b_i \in \mathbb{Z}$, we conclude that $|b_j| = 1$ and the remaining coordinates in b have to be 0, i.e $b_{j'} = 0$ for all $j' \in [n] \setminus \{j\}$.
2. Follows from $3|b_j| = \|b\|^2$ and b being integral.
3. We can re-write the condition $3|b_j| = \|b\|^2$ as $3|b_j| = \sum_{i=1}^n b_i^2$. Rearranging the terms, we have

$$|b_j|\ (3 - |b_j|) = \sum_{i\neq j} b_i^2. \tag{2}$$

If $b_j = \pm 1$, then $\sum_{i\neq j} b_i^2 = 2$. Further, b is integral. Hence, b has exactly 2 other non-zero coordinates b_{j_1}, b_{j_2}, $j \neq j_1, j_2$, such that $|b_{j_1}| = |b_{j_2}| = 1$.

4. We have equation (2). The RHS is a sum of squares and hence the LHS is non-negative. Moreover, b is not all-zeroes vector implies that $b_j \neq 0$. Therefore, $|b_j| \in (0,3] \cap \mathbb{Z}$. If $b_j = \pm 2$, then in order to satisfy $\sum_{i \neq j} b_i^2 = 2$ using integral b_i's, exactly two coordinates b_{j_1}, b_{j_2} should be ± 1, where $j \neq j_1, j_2$. However, in this case, $3|b_{j_1}| = 3|b_{j_2}| = 3 \notin \{0, \|b\|^2 = 6, 3\|b\|^2 = 18\}$, thus contradicting Claim 11. The conclusion follows from parts (2) and (3). ◀

Using the properties of the orthogonal basis B of L_C given in Claims 11 and 12, we show that B is equivalent (up to permutations of its columns) to a block diagonal matrix, i.e

$$B \cong \begin{bmatrix} B_1 & 0 & \cdots & 0 \\ 0 & B_2 & \cdots & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & \cdots & B_k \end{bmatrix}$$

where each B_i is either the 1×1 matrix $[1]$ or the 1×1 matrix $[3]$ or the 4×4 matrix obtained from M by negating a subset of its columns, $\mathcal{T}(M)$. It follows that $L_C \cong \otimes_i L_i$ such that B_i is the basis for the lower dimensional lattice L_i.

Let us pick a row b of B with the smallest support. Fix an index $j \in [n]$ to be the index of a non-zero entry with minimum absolute value in b, i.e. $j = \arg\min_k \{|b_k|\}$. As b is a row of a basis matrix, b cannot be the all-zeroes vector and therefore there exists a $j \in [n]$ such that $|b_j| > 0$. Since we are only interested in equivalence (that allows for permutation of coordinates), we may assume without loss of generality that $j = 1$ by permuting the coordinates. By Claim 11, we have that $3|b_1| \in \{\|b\|^2, 3\|b\|^2\}$. We consider each of these cases separately.

1. Suppose $3|b_1| = 3\|b\|^2$. By Claim 12(1), $b = (\pm 1, 0, \ldots, 0)$. Since B is an orthogonal basis, $\langle b, b' \rangle = 0 \Rightarrow b'_1 = 0$ for all $b' \neq b \in B$. The orthogonality of B therefore forces all other basis vectors to take a value of 0 at the 1^{st} coordinate. Thus B is of the form

$$B = \left(\begin{array}{c|c} \pm 1 & 0 \cdots 0 \\ \hline 0 & \\ \vdots & B' \\ 0 & \end{array}\right).$$

   Therefore, we obtain $L_C \cong \mathbb{Z} \otimes L'$, where L' is an orthogonal $(n-1)$-dimensional lattice generated by the basis matrix restricted to the coordinates other than 1, say, B'. From Claim 6, it follows that $L' = C' + 3\mathbb{Z}^{n-1}$ for some ternary linear code $C' \subseteq \mathbb{F}_3^{n-1}$. Thus L' satisfies the induction hypothesis and we have the desired decomposition.
2. Suppose $3|b_1| = \|b\|^2$. We can re-write this condition as $3|b_1| = \sum_{i=1}^n b_i^2$. Rearranging the terms, we have

$$|b_1| \ (3 - |b_1|) = \sum_{i \neq 1} b_i^2.$$

   Since the RHS is a sum of squares, it should be non-negative.

   (i) If RHS is 0, then $b_1 = \pm 3$ and therefore, it follows from Claim 12(2) that $b = (\pm 3, 0, \ldots, 0)$. The orthogonality of B forces all other basis vectors to take a value of

0 at the 1^{st} coordinate.

$$B = \left(\begin{array}{c|c} \pm 3 & 0 \cdots 0 \\ \hline 0 & \\ \vdots & B' \\ 0 & \end{array}\right)$$

Therefore, we obtain $L_C \cong 3\mathbb{Z} \otimes L'$, where L' is an orthogonal $(n-1)$-dimensional lattice generated by the basis matrix restricted to the coordinates other than 1, say B'. From Claim 6, it follows that $L' = C' + 3\mathbb{Z}^{n-1}$ for some ternary linear code $C' \subseteq \mathbb{F}_3^{n-1}$. Thus L' satisfies the induction hypothesis and we have the desired decomposition.

(ii) If RHS is strictly positive, then $|b_1| \in (0,3) \cap \mathbb{Z} = \{1,2\}$. By Claim 12(4), $b_1 \neq \pm 2$. Therefore, $b_1 = \pm 1$. By Claim 12(3), we have that b has exactly three non-zero coordinates and they are ± 1. By permuting the coordinates of B, we can write $b \equiv (\pm 1, \pm 1, \pm 1, 0, \cdots, 0)$.

Since we picked the row b to be the one with the smallest support, it follows that every row has at least 3 non-zero coordinates. By Claims 11 and 12(1), this is possible only if for every other row b', there exists $j' \in [n]$ such that $3|b'_{j'}| = \|b'\|^2$. By Claim 12(4), every other row b' has all its coordinates in $\{0, \pm 1, \pm 3\}$. By Claim 12(2), every other row b' has none of its coordinates in $\{\pm 3\}$. Therefore, every other row b' has all its coordinates in $\{0, \pm 1\}$. By Claim 12(3), every row of the basis matrix has the same form as b: they have exactly three non-zero entries each of which is ± 1.

Since the rows of the basis matrix are orthogonal, it follows that the basis matrix B is a weighing matrix of order n with weight 3. By Theorem 10, B is obtained from $\otimes_{n/4} M$ by either negating some rows or columns and by interchanging rows or columns. We recall that interchanging or negating the rows of the basis matrix of a lattice preserves the basis property while interchanging columns is equivalent to permuting the coordinates. Hence $L_C = L(B) \cong \otimes_{i=1}^{n/4} L(\mathcal{T}_i(M))$, where each $\mathcal{T}_i(M)$ is a 4×4 matrix obtained by negating a subset of columns of M.

(2) $\equiv$ (3): We now show that L_C decomposes into direct product of lower dimensional lattices, $L_C \cong \otimes_i L_i$ if and only if the code C also decomposes, $C \cong \otimes_i C_i$.

Let $L_C \cong \otimes_i L_i$. Without loss of generality, we can consider $L_C = \otimes_i L_i$. We have $C = L_C \bmod 3 = \otimes_i L_i \bmod 3$. We observe that if L_i has dimension n_i, then $L_i \supseteq 3\mathbb{Z}^{n_i}$. Therefore, $C_i = L_i \bmod 3$ is a ternary code. Let $C_i := L_i \bmod 3$ for every i. Then $C = \otimes_i C_i$. (If $c \in C$, then $c \in L$ and hence the projection of c to the subset of coordinates corresponding to L_i is in C_i. Let $c_i \in C_i$ for every i. The concatenated vector $\otimes_i c_i$ is in $\otimes_i L_i \mod 3$ and hence is in C.)

To show the other side, let $C \cong \otimes_i C_i$, where each $C_i \subseteq \mathbb{F}_3^{n_i}$ and $n = \sum_i n_i$. Therefore $L_C = C + 3\mathbb{Z}^n \cong \otimes_i C_i + 3\mathbb{Z}^n \cong \otimes_i (C_i + 3\mathbb{Z}^{n_i})$, since $\mathbb{Z}^n \cong \otimes_i \mathbb{Z}^{n_i}$. ◀

3.2 The algorithm

Theorem 3 shows that a lattice of the form $C + 3\mathbb{Z}^n$ is orthogonal if and only if the underlying code decomposes into direct product of ternary linear codes isomorphic to $\{0,1,2\}$ or $\{0\}$ or the four dimensional code generated by $\mathcal{T}(M) \bmod 3$, where $\mathcal{T}(M)$ is obtained from matrix M by negating a subset of its columns. We now give a polynomial time algorithm which finds the decomposition of the code C into the component codes, C_i, if there exists one.

Algorithm 1 decompose – length – 1(G):

Input: $G = \{g_1, \ldots, g_n\} \in \mathbb{F}_3^n$ (A generator for the code C)

1: **for** $j \in \{1, \cdots, n\}$ **do**
2: Let $G' \leftarrow$ projection of vectors in G on coordinates $[n] \setminus \{j\}$
3: For $g \in G'$, define $g^0, g^1, g^2 \in \mathbb{F}_3^n$ as the n-dimensional vectors obtained by extending g using 0, 1 and 2 along the j'th coordinate respectively.
4: **if** $g^0,\ g^1,\ g^2 \in C$ for all $g \in G'$ **then**
5: **return** j
6: **return** FAIL

Therefore, if the lattice L_C is orthogonal, the algorithm decides in polynomial time if it is orthogonal and also gives the orthogonal basis for the lattice.

The algorithm recursively finds the component codes. If it is unable to decompose the code at any stage, then it declares that L_C is not orthogonal. At every step we check if $C \cong \{0,1,2\} \times C'$ or $\{0\} \times C'$ or $C_{\mathcal{T}(M)} \times C'$ where $C_{\mathcal{T}(M)}$ is the code generated by $\mathcal{T}(M)$ mod 3 and then recurse on C'.

Proof of Theorem 4. Given a basis for L_C as input, we first compute the generator for C. From Theorem 3, we know that if L_C is orthogonal, then $C \cong \otimes_i C_i$ where each C_i is either the length-1 code $\{0,1,2\}$ or the length-1 code $\{0\}$ or a 4-dimensional code generated by the rows of $\mathcal{T}(M)$ mod 3 where $\mathcal{T}(M)$ obtained from matrix M by negating a subset of its columns.

The algorithm therefore in each step decides if $C \cong \{0,1,2\} \otimes C'$ or $C \cong \{0\} \otimes C'$ or $C \cong C_{\mathcal{T}(M)} \otimes C'$, where $C_{\mathcal{T}(M)}$ denotes the code generated by $\mathcal{T}(M)$ mod 3. Theorem 13 shows that using Algorithm 1 we can check in $O(n^4)$ time, if $C \cong \{0,1,2\} \otimes C'$. The same algorithm can be modified to check in $O(n^4)$ time, if $C \cong \{0\} \otimes C'$. Theorem 14 shows that Algorithm 2 can verify if $C \cong C_{\mathcal{T}(M)} \otimes C'$ in $O(n^7)$ time. If any one of the algorithms finds a decomposition, then we recurse in the lower dimensional code C' to find further decomposition. We recurse at most n times. If all the algorithms fail to find a decomposition, then L_C is not orthogonal. Therefore, it takes $O(n^8)$ time to decide if L_C is orthogonal. ◀

We now describe the individual algorithms to verify if $C \cong \{0,1,2\} \otimes C'$ or $C \cong \{0\} \otimes C'$ or $C \cong C_{\mathcal{T}(M)} \otimes C'$.

▶ **Theorem 13.** *Let C be a ternary linear code and $G = \{g_1, \ldots, g_n\} \in \mathbb{F}_3^{n \times n}$ be its generator. Then Algorithm 1 decides if $C \cong \{0,1,2\} \otimes C'$ for some linear code $C' \subseteq \mathbb{F}_3^{n-1}$ and if so outputs the coordinate corresponding to the direct product decomposition. Moreover the algorithm runs in time $O(n^4)$.*

Proof. For $j \in [n]$, let $C'_{\overline{j}} \subseteq \mathbb{F}_3^{n-1}$ be the projection of C on the indices $[n] \setminus \{j\}$ and for a vector $c \in C'_{\overline{j}}$, let $c^0, c^1, c^2 \in \mathbb{F}_3^n$ be extensions of c using $0, 1, 2$ respectively along the j'th coordinate. We note that $C \cong \{0,1,2\} \otimes C'$ for some ternary linear code C' if and only if there exists an index $j \in [n]$, such that

$$C = \left\{ c^0, c^1, c^2 \mid \ \forall\ c \in C'_{\overline{j}} \right\}. \tag{3}$$

From the definition of $C'_{\overline{j}}$, it follows that $C \subseteq \{c^0, c^1, c^2 \mid \ \forall\ c \in C'_{\overline{j}}\}$ up to a permutation of coordinates. So, the algorithm just needs to verify if the other side of the containment holds for some j.

Algorithm 2 decompose – length – 4(G):

Input: $G \in \mathbb{F}_3^{n \times n}$ (Generator for C)

for $j_1, j_2, j_3, j_4 \in \{1, 2, \cdots, n\}$ **do**
  Let $G' \leftarrow$ projection of vectors in G on coordinates $[n] \setminus \{j_1, j_2, j_3, j_4\}$
  Let $G'' \leftarrow$ projection of vectors in G on coordinates $\{j_1, j_2, j_3, j_4\}$
  for $S \subseteq [4]$ **do**
    Let $\mathcal{T}(M) \leftarrow M$ with columns in S negated
    if $C_{\mathcal{T}(M)} \equiv$ Code generated by G'' **then**
      For $g \in G'$ define $g^{p_1}, g^{p_2}, g^{p_3}, g^{p_4} \in \mathbb{F}_3^n$ be n-dimensional vectors obtained by extending g using the rows of $\mathcal{T}(M)$ along the j_1, j_2, j_3, j_4 coordinates.
      if $g^{p_1}, g^{p_2}, g^{p_3}, g^{p_4} \in C$ for all $g \in G'$ **then**
        return j_1, j_2, j_3, j_4 and $\mathcal{T}(M)$
return FAIL

Let G' be the set of vectors of G projected on the coordinates $[n] \setminus \{j\}$. Algorithm 1 verifies if g^0, g^1 and g^2 are codewords in C, for every vector $g \in G'$. We now show that this is sufficient. Since C is a code, if g^0, g^1, $g^2 \in C$ for every $g \in G'$, then all linear combinations of these vectors are also in C. Therefore, $\{c^0, c^1, c^2 \mid \forall\ c \in C'_{\bar{j}}\} \subseteq C$.

It takes $O(n^2)$ time to compute a parity check matrix from the generator G and $O(n^2)$ time to verify if an input vector is a codeword using the parity check matrix. For every possible choice of the index j, Algorithm 1 checks if each of the $3n$ vectors of the form g^0, g^1, g^2 are C. Therefore, Algorithm 1 takes $O(n^4)$ time to decide if $C \cong \{0, 1, 2\} \otimes C'$. ◀

▶ **Theorem 14.** *Let C be a ternary linear code and $G = \{g_1, \ldots, g_n\} \in \mathbb{F}_3^{n \times n}$ be its generator. For a matrix $\mathcal{T}(M)$ obtained by negating a subset of columns of M, let $C_{\mathcal{T}(M)}$ be the length-4 code whose generators are the rows of $\mathcal{T}(M)$. Then Algorithm 2 decides if $C \cong C_{\mathcal{T}(M)} \otimes C'$ for some linear codes $C' \subseteq \mathbb{F}_3^{n-4}$ and $C_{\mathcal{T}(M)} \subseteq \mathbb{F}_3^4$ and if so outputs the coordinates corresponding to the direct product decomposition as well as the matrix $\mathcal{T}(M)$. Moreover the algorithm runs in time $O(n^7)$.*

Proof. For $1 \leq j_1 < j_2 < j_3 < j_4 \leq n$, let C''_{j_1,j_2,j_3,j_4} be the projection of C on the indices $\{j_1, j_2, j_3, j_4\}$. We first verify if C''_{j_1,j_2,j_3,j_4} is the code generated by the rows of $\mathcal{T}(M)$ (denoted as $C_{\mathcal{T}(M)}$) for some $\mathcal{T}(M)$ which is obtained by negating a subset of columns of M. We would like to check if every $c \in C''_{j_1,j_2,j_3,j_4}$ is in $C_{\mathcal{T}(M)}$ and vice versa. For this purpose, it is sufficient to check if the generator vectors of C''_{j_1,j_2,j_3,j_4} are codewords in $C_{\mathcal{T}(M)}$ and each row of $\mathcal{T}(M)$ is a codeword in C''_{j_1,j_2,j_3,j_4}. We know that the generators of C''_{j_1,j_2,j_3,j_4} are contained in G'' where G'' is the set of vectors in G projected on the indices $\{j_1, j_2, j_3, j_4\}$.

Once we fix $\mathcal{T}(M)$ such that $C''_{j_1,j_2,j_3,j_4} = C_{\mathcal{T}(M)}$, to see if $C \cong C_{\mathcal{T}(M)} \otimes C'$ for some ternary linear code $C' \subseteq \mathbb{F}_3^{n-4}$. Define $C'_{\bar{j_1},\bar{j_2},\bar{j_3},\bar{j_4}}$ to be the projection of C on the indices $[n] \setminus \{j_1, j_2, j_3, j_4\}$. For a vector $c \in C'_{\bar{j_1},\bar{j_2},\bar{j_3},\bar{j_4}}$, let $c^p \in \mathbb{F}_3^n$ be the extensions of c using a codeword $p \in C_{\mathcal{T}(M)}$ along the j_1, j_2, j_3, j_4 coordinates. We note that $C \cong C_{\mathcal{T}(M)} \otimes C'$ for some ternary linear code C' if and only if there exist indices $j_1, j_2, j_3, j_4 \in [n]$, such that

$$C = \left\{ c^p \mid c \in C'_{\bar{j_1},\bar{j_2},\bar{j_3},\bar{j_4}}, p \in C_{\mathcal{T}(M)} \right\}. \tag{4}$$

From the definition of $C'_{\bar{j_1},\bar{j_2},\bar{j_3},\bar{j_4}}$ and C''_{j_1,j_2,j_3,j_4} $(= C_{\mathcal{T}(M)})$, it follows that $C \subseteq \{c^p \mid c \in C'_{\bar{j_1},\bar{j_2},\bar{j_3},\bar{j_4}}, p \in C_{\mathcal{T}(M)}\}$. So, the algorithm just needs to verify if the other side of the containment holds for some indices j_1, j_2, j_3, j_4.

Let G' be the set of vectors of G projected on the coordinates $[n] \setminus \{j_1, j_2, j_3, j_4\}$. Algorithm 2 verifies if g^{p_0}, g^{p_1}, g^{p_3} and g^{p_4} are codewords in C, for every vector $g \in G'$. We now show that this is sufficient. Since C is a code, if $g^{p_0}, g^{p_1}, g^{p_3}, g^{p_4} \in C$ for every $g \in G'$ and $p_i \in \mathcal{T}(M)$, then all linear combinations of these vectors are also in C. Therefore, $\{c^p \mid c \in C'_{\bar{j_1},\bar{j_2},\bar{j_3},\bar{j_4}}, p \in C_{\mathcal{T}(M)}\} \subseteq C$.

There are $2^4 4^4$ possible choices of $\mathcal{T}(M)$ including permutations. For each matrix $\mathcal{T}(M)$, it takes $O(n)$ time to verify if $C_{\mathcal{T}(M)} = C''_{j_1,j_2,j_3,j_4}$. As we had seen that it takes $O(n^2)$ time to verify if an input vector is a codeword using the parity check matrix. We perform this check for $4n$ vectors of the form $\{g^{p_0}, g^{p_1}, g^{p_3}, g^{p_4} \mid g \in G'\}$. So, for a given $\mathcal{T}(M)$ such that $C_{\mathcal{T}(M)} = C''_{j_1,j_2,j_3,j_4}$, It takes $O(n^3)$ time to verify $C \cong C_{\mathcal{T}(M)} \otimes C'$.

Therefore, for every possible choice of $\{j_1, j_2, j_3, j_4\}$, Algorithm 2 takes $O(n^3)$ time to verify if $C \cong C_{\mathcal{T}(M)} \otimes C'$. Since there are at most $\binom{n}{4}$ possible choices of indices, it takes $O(n^7)$ time in total to decide if $C \cong C_{\mathcal{T}(M)} \otimes C'$. ◀

Acknowledgments. We thank Daniel Dadush for helpful suggestions and pointers.

References

1. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.
2. Laszlo Babai. Automorphism groups, isomorphism, reconstruction. In *Handbook of Combinatorics*, volume chapter 27, pages 1447–1540. North-Holland, 1996.
3. H.C. Chan, C.A. Rodger, and J. Seberry. On inequivalent weighing matrices. *Ars Combinatoria*, 21(A):229–333, 1986.
4. Karthekeyan Chandrasekaran, Venkata Gandikota, and Elena Grigorescu. Deciding Orthogonality in Construction-A Lattices. Under Preparation, 2015.
5. John H. Conway and Neil J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer-Verlag, New York, 1998.
6. Ishay Haviv and Oded Regev. On the lattice isomorphism problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 391–404, 2014.
7. Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 193–206, 1983.
8. Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.
9. Hendrik W. Lenstra and Alice Silverberg. Lattices with symmetries. Manuscript, 2014.
10. Hendrik W. Lenstra and Alice Silverberg. Revisiting the gentry-szydlo algorithm. In *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 280–296. Springer Berlin Heidelberg, 2014.
11. Daniele Micciancio. Lecture notes on lattice algorithms and applications, Winter 2010. Lecture 2.
12. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, 2009.
13. Wilhelm Plesken and Bernd Souvignier. Computing isometries of lattices. *J. Symb. Comput.*, 24(3/4):327–334, 1997.

14 Claus-Peter Schnorr. Factoring integers by CVP algorithms. In *Number Theory and Cryptography – Papers in Honor of Johannes Buchmann on the Occasion of His 60th Birthday*, pages 73–93, 2013.

15 Mathieu Dutour Sikiric, Achill Schürmann, and Frank Vallentin. Complexity and algorithms for computing voronoi cells of lattices. *Math. Comput.*, 78(267):1713–1731, 2009.

Ordered Tree-Pushdown Systems

Lorenzo Clemente[*1], Paweł Parys[†1], Sylvain Salvati[2], and Igor Walukiewicz[‡2]

1 **University of Warsaw**
Warsaw, Poland
2 **CNRS, Université de Bordeaux, INRIA**
Bordeaux, France

Abstract

We define a new class of pushdown systems where the pushdown is a tree instead of a word. We allow a limited form of lookahead on the pushdown conforming to a certain ordering restriction, and we show that the resulting class enjoys a decidable reachability problem. This follows from a preservation of recognizability result for the backward reachability relation of such systems. As an application, we show that our simple model can encode several formalisms generalizing pushdown systems, such as ordered multi-pushdown systems, annotated higher-order pushdown systems, the Krivine machine, and ordered annotated multi-pushdown systems. In each case, our procedure yields tight complexity.

1998 ACM Subject Classification D.1.1 [Model checking]: Software/Program Verification, D.2.4 [Applicative (Functional) Programming]: Programming techniques, F.1.1 [Automata]: Models of Computation, F.3.1 [Specifying and Verifying and Reasoning about Programs]: Mechanical verification, F.4.1 [Lambda calculus and related systems]: Mathematical Logic

Keywords and phrases reachability analysis, saturation technique, pushdown automata, ordered pushdown automata, higher-order pushdown automata, higher-order recursive schemes, simply-typed lambda calculus, Krivine machine

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.163

1 Introduction

Context. Modeling complex systems requires to strike the right balance between the accuracy of the model, and the complexity of its analysis. A successful example is given by *pushdown systems*, which are a popular class of infinite-state systems arising in diverse contexts, such as language processing, data-flow analysis, security, computational biology, and program verification. Many interesting analyses reduce to checking reachability in pushdown systems, which can be decided in PTIME using, e.g., the popular *saturation technique* [5, 14] (cf. also the recent survey [10]). Pushdown systems have been generalized in several directions. One of them are *tree-pushdown systems* [15], where the pushdown is a tree instead of a word. Unlike for ordinary pushdown systems, non-destructive lookahead on the tree pushdown leads to undecidability. In this work we propose an ordering condition permitting a limited non-destructive lookahead on a tree pushdown.

* This work was partially supported by the National Science Center (decision DEC-2013/09/B/ST6/01575).
† This work was partially supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).
‡ This work was partially supported by the Technische Universität München – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme, grant n. 291763.

A seemingly unrelated generalization is *ordered multi-pushdown systems* [6, 3, 2], where several linear pushdowns are available instead of just one. Since already two unrestricted linear pushdowns can simulate a Turing machine, an ordering restriction is put on popping transitions, requiring that all pushdowns smaller than the popped one are empty. Reachability in this model is 2-EXPTIMEc [3].

Higher-order pushdown systems provide another type of generalization. Here pushdowns can be nested inside other pushdowns [23, 20]. *Collapsible pushdown systems* [21, 17] additionally enrich pushdown symbols with *collapse links* to inner sub-pushdowns. This allows the automaton to push a new symbol and to save, at the same time, the current context in which the symbol is pushed, and to later return to this context via a collapse operation. *Annotated pushdown systems* [7] (cf. also [19]) provide a simplification of collapsible pushdown systems by replacing collapse links with arbitrary pushdown annotations[1]. The *Krivine machine* [24] is a related model which evaluates terms in simply-typed λY-calculus. Reachability in all these models is $(\mathsf{n}-1)$-EXPTIMEc [7, 24] (where n is the order of nesting pushdowns/functional parameters), and one exponential higher in the presence of alternation. Even more general, *ordered annotated multi-pushdown systems* [16] have several annotated pushdown systems under an ordering restriction similar to [3] in the first-order case. They subsume both ordered multi-pushdown systems and annotated pushdown systems. The saturation method (cf. [10]) has been adapted to most of these models, and it is the basis of the prominent MOPED tool [13] for the analysis of pushdown systems, as well as the C-SHORe model-checker for annotated pushdown systems [8].

Contributions. Motivated by a unification of the results above, we introduce *ordered tree-pushdown systems*. These are tree-pushdown systems with a limited destructive lookahead on the pushdown. We introduce an order between pushdown symbols, and we require that, whenever a sub-pushdown is read, all sub-pushdowns of smaller order must be discarded. The obtained model is expressive enough to simulate all the systems mentioned above, and is still not Turing-powerful thanks to the ordering condition. Our contributions are:

- **(i)** A general preservation of recognizability result for ordered tree-pushdown systems.
- **(ii)** A conceptually simple saturation algorithm working on finite tree automata representing sets of configurations (instead of more ad-hoc automata models), subsuming and unifying previous constructions.
- **(iii)** A short and simple correctness proof.
- **(iv)** Direct encodings of several popular extensions of pushdown systems, such as ordered multi-pushdown systems, annotated pushdown systems, the Krivine machine, and ordered annotated multi-pushdown systems.
- **(v)** Encoding of our model into Krivine machines with states, that in turn are equivalent to collapsible pushdown automata.
- **(vi)** A complete complexity characterization of reachability in ordered tree-pushdown systems and natural subclasses thereof.

Related work. Our work can be seen as a generalization of the saturation method for collapsible pushdown automata [7] to a broader class of rewriting systems. This method has

[1] Collapsible and annotated systems generate the same configuration graphs when started from the same initial configuration, since new annotations can only be created to sub-pushdowns of the current pushdown. However, annotated pushdown systems have a richer backward reachability set which includes non-constructible pushdowns.

been already generalized in [16] to multi-stack higher-order systems; in particular for ordered, phase-bounded, and scope-bounded restrictions. Another related work is a saturation method for recursive program schemes [9]. Schemes are equivalent to λY-calculus, so our formalism can be used to obtain a saturation method for schemes.

Ordered tree-pushdown systems proposed in the present paper unify these approaches. The encodings of the above mentioned systems are direct and work step-to-step. By contrast, the encoding of the Krivine machine to higher-order pushdowns is rather sophisticated [17, 26], and even more so its proof of correctness. The converse encoding of annotated higher-order pushdowns into Krivine machines is conceptually easier, but technically quite long for at least two reasons: a state has to be encoded by a tuple of terms, and transitions of the automaton need to be implemented with beta-reduction.

Concerning multi-pushdown systems, there exist restrictions that we do not cover in this paper. In [16] decidability is proved for annotated multi-pushdowns with phase-bounded and scope-bounded restrictions. For standard multi-pushdown systems, split-width has been proposed as a unifying restriction [12].

Outline. In Sec. 2 we introduce common notions. In Sec. 3 we define our model and we present our saturation-based algorithm to decide reachability. In Sec. 4 we show that ordered systems can optimally encode several popular formalisms. In Sec. 5 we discuss the notion of safety from the Krivine machine and higher-order pushdown automata, and how it relates to our model. In Sec. 6 we conclude with some perspectives on open problems. Full proofs can be found in the technical report [11].

2 Preliminaries

We work with rewriting systems on ranked trees, and with alternating tree automata. The novelty is that every letter of the ranked alphabet will have an order. A tree has the order determined by the letter in the root. The order itself is used to constrain rewriting rules.

An *alternating transition system* is a tuple $\mathcal{S} = \langle \mathcal{C}, \rightarrow \rangle$, where $\mathcal{C}$ is the set of configurations and $\rightarrow \subseteq \mathcal{C} \times 2^{\mathcal{C}}$ is the alternating transition relation. For two sets of configurations $A, B \subseteq \mathcal{C}$ we define $A \rightarrow_1 B$ iff, for every $c \in A$, either $c \in B$, or there exists $C \subseteq B$ s.t. $c \rightarrow C$, and we denote by $\rightarrow_1^*$ its reflexive and transitive closure. The set of *predecessors* of a set of configurations $C \subseteq \mathcal{C}$ is $\mathrm{Pre}^*(C) = \{c \mid \{c\} \rightarrow_1^* C\}$.

Ranked trees. Let $\mathbb{N}$ be the set of non-negative integers, and let $\mathbb{N}_{>0}$ be the set of strictly positive integers. A *node* is an element $u \in \mathbb{N}_{>0}^*$. A node u is a *child* of a node v if $u = v \cdot i$ for some $i \in \mathbb{N}_{>0}$. A *tree domain* is a non-empty prefix-closed set of nodes $D \subseteq \mathbb{N}_{>0}^*$ s.t., if $u \cdot (i+1) \in D$, then $u \cdot i \in D$ for every $i \in \mathbb{N}_{>0}$. A *leaf* is a node u in D without children. A *ranked alphabet* is a pair (Σ, rank) of a set of symbols Σ together with a ranking function $\mathsf{rank} : \Sigma \to \mathbb{N}$. A Σ-*tree* is a function $t : D \to \Sigma$, where D is a tree domain, s.t., for every node u in D labelled with a symbol $t(u)$ of rank k, u has precisely k children. For a Σ-tree $t : D \to \Sigma$ and a label $a \in \Sigma$, let $t^{-1}(a) = \{u \in D \mid t(u) = a\}$ be the set of nodes labelled with a. For a tree t and a node u therein, the *subtree* of t at u is defined as expected. We denote by $\mathcal{T}(\Sigma)$ the set of Σ-trees.

Order of a tree. In this paper we will give a restriction on a tree rewriting system guaranteeing that $\mathrm{Pre}^*(C)$ is regular for every regular set C. This restriction will use the notion of an order of a tree. The order of a tree is simply determined by the order of the symbol in

the root. Therefore, we suppose that our alphabet Σ comes with a function $\mathsf{ord} : \Sigma \to \mathbb{N}$. The *order* of a tree t is $\mathsf{ord}(t) := \mathsf{ord}(t(\varepsilon))$.

Rewriting. Let $\mathcal{V}_0, \mathcal{V}_1, \dots$ be pairwise disjoint infinite sets of variables; and let $\mathcal{V} = \bigcup_n \mathcal{V}_n$. We consider the extended alphabet $\Sigma \cup \mathcal{V}$ where a variable $\mathsf{x} \in \mathcal{V}_n$ has rank 0 and order n. We will work with the set $\mathcal{T}(\Sigma, \mathcal{V})$ of $(\Sigma \cup \mathcal{V})$-trees. For such a tree t, let $\mathcal{V}(t)$ be the set of variables appearing in it. We say that t is *linear* if each variable in $\mathcal{V}(t)$ appears exactly once in t. For some $(\Sigma \cup \mathcal{V})$-tree u, t is *u-ground* if $\mathcal{V}(t) \cap \mathcal{V}(u) = \varnothing$. A *substitution* is a finite partial mapping $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma \cup \mathcal{V})$ respecting orders, i.e., $\mathsf{ord}(\sigma(\mathsf{x})) = \mathsf{ord}(\mathsf{x})$. Given a $(\Sigma \cup \mathcal{V})$-tree t and a substitution σ, $t\sigma$ is the $(\Sigma \cup \mathcal{V})$-tree obtained by replacing each variable x in t in the domain of σ with $\sigma(\mathsf{x})$. A *rewrite rule* over Σ is a pair $l \to r$ of $(\Sigma \cup \mathcal{V})$-trees l and r s.t. $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ and l is linear.[2]

Alternating tree automata. An *alternating tree automaton* (or just *tree automaton*) is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta \rangle$ where Σ is a finite ranked alphabet, Q is a finite set of states, and $\Delta \subseteq Q \times \Sigma \times (2^Q)^*$ is a set of alternating transitions of the form $p \xrightarrow{a} P_1 \cdots P_n$, with a of rank n. We say that $\mathcal{A}$ is *non-deterministic* if, for every transition as above, all P_j's are singletons, and we omit the braces in this case. An automaton is *ordered* if, for every state p and symbols a, b s.t. $p \xrightarrow{a} \cdots$ and $p \xrightarrow{b} \cdots$, we have $\mathsf{ord}(a) = \mathsf{ord}(b)$. We assume w.l.o.g. that automata are ordered, and we denote by $\mathsf{ord}(p)$ the order of state p. The transition relation is extended to a set of states $P \subseteq Q$ by defining $P \xrightarrow{a} P_1 \cdots P_n$ iff, for every $p \in P$, there exists a transition $p \xrightarrow{a} P_1^p \cdots P_n^p$, and $P_j = \bigcup_{p \in P} P_j^p$ for every $j \in \{1, \dots, n\}$. It will be useful later in the definition of the saturation procedure to define run trees not just on ground trees, but also on trees possibly containing variables. A variable of order k is treated like a leaf symbol which is accepted by all states of the same order. Let $P \subseteq Q$ be a set of states, and let $t : D \to (\Sigma \cup \mathcal{V})$ be an input tree. A *run tree from P on t* is a 2^Q-tree[3] $s : D \to 2^Q$ over the same tree domain D s.t. $s(\varepsilon) = P$, and:

- **(i)** if $t(u) = a$ is not a variable and of rank n, then $s(u) \xrightarrow{a} s(u \cdot 1) \cdots s(u \cdot n)$, and
- **(ii)** if $t(u) = \mathsf{x}$ then $\forall p \in s(u), \mathsf{ord}(p) = \mathsf{ord}(\mathsf{x})$.

The *language* recognized by a set of states $P \subseteq Q$, denoted by $\mathcal{L}(P)$, is the set of Σ-trees t s.t. there exists a run tree from P on t.

3 Ordered tree-pushdown systems

We introduce a generalization of pushdown systems, where the pushdown is a tree instead of a word. An *alternating ordered tree-pushdown system* (AOTPS) of order $n \in \mathbb{N}_{>0}$ is a tuple $\mathcal{S} = \langle n, \Sigma, P, \mathcal{R} \rangle$ where Σ is an ordered alphabet containing symbols of order at most n, P is a finite set of *control locations*, and $\mathcal{R}$ is a set of rules of the form $p, l \to S, r$ s.t. $p \in P$ and $S \subseteq P$. Moreover, $l \to r$ is a rewrite rule over Σ of one of the two forms:

$$\textit{(shallow)}\colon\ a(u_1, \dots, u_m) \to r \quad \text{or} \quad \textit{(deep)}\colon\ a(u_1, \dots, u_k, b(v_1, \dots, v_{m'}), u_{k+1}, \dots, u_m) \to r$$

[2] Notice that we require that all the variables appearing on the r.h.s. r also appear on the l.h.s. l. All our results carry over even by allowing some variables on the r.h.s. r not to appear on the l.h.s. l, but we forbid this for simplicity of presentation.

[3] Strictly speaking 2^Q does not have a rank/order. It is easy to duplicate each subset at every rank/order to obtain an ordered alphabet, which we avoid for simplicity.

where each u_i, v_j is either r-ground or a variable, and for the second form we require

(ordering condition): if $\mathsf{ord}(u_i) \leqslant \mathsf{ord}(b)$, then u_i is r-ground; for $i = 1, \ldots, m$.

The rules in $\mathcal{R}$ where $l \to r$ is of the first form are called *shallow*, the others are *deep*. The tree $b(v_1, \ldots, v_{m'})$ in a deep rule is called the *lookahead subtree* of l. A rule $l \to r$ is *flat* if each u_i, v_j is just a variable. Let $\mathcal{R}_{\mathsf{ord}(b)}$ be the set of deep rules, where the lookahead symbol b is of order $\mathsf{ord}(b)$. For example, $a(\mathsf{x}, \mathsf{y}) \to c(a(\mathsf{x}, \mathsf{y}), \mathsf{x})$ is shallow and flat, but $a(b(\mathsf{x}), \mathsf{y}) \to c(\mathsf{x}, \mathsf{y})$ is deep (and flat); here necessarily $\mathsf{ord}(\mathsf{y}) > \mathsf{ord}(b)$. Finally, $a(c, d, \mathsf{x}) \to b(\mathsf{x})$ is not flat since c and d are not variables. In Sec. 4 we provide more examples of such rewrite rules by encoding many popular formalisms. While l must be linear, r may be non-linear, thus sub-trees can be duplicated. The *size* of $\mathcal{S}$ is $|\mathcal{S}| := |\Sigma| + |P| + |\mathcal{R}|$, where $|\mathcal{R}| := \sum_{(p,l \to S,r) \in \mathcal{R}} (1 + |l| + |S| + |r|)$.

Rewrite rules induce an alternating transition system $\langle \mathcal{C}_\mathcal{S}, \to_\mathcal{S} \rangle$ by root rewriting. The set of configurations $\mathcal{C}_\mathcal{S}$ consists of pairs (p, t) with $p \in P$ and $t \in \mathcal{T}(\Sigma)$, and, for every configuration (p, t), set of control locations $S \subseteq P$, and tree u, $(p, t) \to_\mathcal{S} S \times \{u\}$ if there exists a rule $((p, l) \to (S, r)) \in \mathcal{R}$ and a substitution σ s.t. $t = l\sigma$ and $u = r\sigma$.

Let $\mathcal{A} = \langle \Sigma, Q, \Delta \rangle$ be a tree automaton s.t. $P \subseteq Q$. The *language of configurations* recognized by $\mathcal{A}$ from P is $\mathcal{L}(\mathcal{A}, P) := \{(p, t) \in \mathcal{C} \mid p \in P \text{ and } t \in \mathcal{L}(p)\}$. Given an initial configuration $(p_0, t_0) \in \mathcal{C}$ and a tree automaton $\mathcal{A}$ recognizing a regular set of target configurations $\mathcal{L}(\mathcal{A}, P) \subseteq \mathcal{C}$, the *reachability problem* for $\mathcal{S}$ amounts to determining whether $(p_0, t_0) \in \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.

3.1 Reachability analysis

We present a saturation-based procedure to decide reachability in AOTPSs. This also shows that backward reachability relation preserves regularity.

▶ **Theorem 3.1** (Preservation of recognizability). *Let $\mathcal{S}$ be an order-n AOTPS and let C be regular set of configurations. Then, $\mathrm{Pre}^*(C)$ is effectively regular, and an automaton recognizing it can be built in n-fold exponential time.*

Let $\mathcal{S} = \langle n, \Sigma, P, \mathcal{R} \rangle$ be an AOTPS. The target set C is given as a tree automaton $\mathcal{A} = \langle \Sigma, Q, \Delta \rangle$ s.t. $\mathcal{L}(\mathcal{A}, P) = C$. W.l.o.g. we assume that in $\mathcal{A}$ initial states (states in P) have no incoming transitions. Classical saturation algorithms for pushdown automata proceed by adding transitions to the original automaton $\mathcal{A}$, until no more new transitions can be added. Here, due to the lookahead of the l.h.s. of deep rules, we need to also add new states to the automaton. However, the total number of new states is bounded once the order of the AOTPS is fixed, which guarantees termination. We construct a tree automaton $\mathcal{B} = \langle \Sigma, Q', \Delta' \rangle$ recognizing $\mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$, where Q' is obtained by adding states to Q, and Δ' by adding transitions to Δ, according to a saturation procedure described below.

For every rule $(p, l \to S, r) \in \mathcal{R}$ and for every subtree v of l we create a new state p^v of the same order as v recognizing all Σ-trees that can be obtained by replacing variables in v by arbitrary trees, i.e., $\mathcal{L}(p^v) = \{v\sigma \mid \sigma : \mathcal{V} \to \mathcal{T}(\Sigma),\ v\sigma \in \mathcal{T}(\Sigma)\}$; recall that the substitution should respect the order. Let Q_0 be the set of such p^v's, and let Δ_0 contain the required transitions. Notice that $|Q_0|, |\Delta_0| \leqslant |\mathcal{R}|$.

In order to deal with deep rules we add new states in the following stratified way. Let $Q'_{n+1} = Q \cup Q_0$. We define sets $Q'_n, \ldots, Q'_1$ inductively starting with Q'_n. Assume that Q'_{i+1} is already defined. We make Q'_i contain Q'_{i+1}. Then we add to Q'_i states for every deep rule $g \in \mathcal{R}_i$ of the form $p, a(u_1, \ldots, u_k, b(\ldots), u_{k+1}, \ldots, u_m) \to S, r$, with $\mathsf{ord}(b) = i$. For simplicity of notation, let us suppose that $u_1, \ldots, u_k$ are of order at most $\mathsf{ord}(b)$, and that

$u_{k+1}, \ldots, u_m$ are of order strictly greater than $\mathsf{ord}(b)$[4]. We add to Q'_i states:

$$(g, P_{k+1}, \ldots, P_m) \in Q'_i \qquad \text{for all } P_{k+1}, \ldots, P_m \subseteq Q'_{i+1}.$$

In particular, to Q_n we add states of the form (g) since n is the maximal order. We define the set of states in $\mathcal{B}$ to be $Q' := Q'_1$.

We add transitions to $\mathcal{B}$ in an iterative process until no more transitions can be added. During the saturation process, we maintain the following invariant: *For $1 \leqslant i \leqslant n$, states in $Q'_i \backslash Q'_{i+1}$ recognize only trees of order i.* Therefore, $\mathcal{B}$ is also an ordered tree automaton. Formally, Δ' is the least set containing $\Delta \cup \Delta_0$ and closed under adding transitions according to the following procedure. Take a deep rule

$$g = (p, a(u_1, \ldots, u_k, b(v_1, \ldots, v_{m'}), u_{k+1}, \ldots, u_m) \to S, r) \in \mathcal{R}_{\mathsf{ord}(b)}$$

and assume as before that the order of u_j is at most $\mathsf{ord}(b)$ for $j \leqslant k$, and strictly bigger than $\mathsf{ord}(b)$ otherwise. We consider a run tree t from S on r in $\mathcal{B}$. For every $j = 1, \ldots, m$ we set: $P^t_j = \{p^{u_j}\}$ if u_j is r-ground, and $P^t_j = \bigcup t(r^{-1}(\mathsf{x}))$ if $u_j = \mathsf{x}$ is a variable appearing in r. The set $\bigcup t(r^{-1}(\mathsf{x}))$ collects all states of $\mathcal{B}$ from which the subtree for which x can be replaced must be accepted. Moreover, for the lookahead subtree $b(v_1, \ldots, v_{m'})$, we let $P^t_b = \{(g, P^t_{k+1}, \ldots, P^t_m)\}$. Analogously, we define $S^t_1, \ldots, S^t_{m'}$ considering $v_1, \ldots, v_{m'}$ instead of $u_1, \ldots, u_m$. Then, we add two transitions:

$$p \xrightarrow{a} P^t_1 \cdots P^t_k P^t_b P^t_{k+1} \cdots P^t_m \qquad \text{and} \qquad (g, P^t_{k+1}, \ldots, P^t_m) \xrightarrow{b} S^t_1 \ldots S^t_{m'} \ . \tag{1}$$

Thanks to the ordering condition, $P^t_{k+1}, \ldots, P^t_m \subseteq Q'_{\mathsf{ord}(b)+1}$, so $(g, P^t_{k+1}, \ldots, P^t_m)$ is indeed a state in $Q'_{\mathsf{ord}(b)}$. For a shallow rule g the procedure is the same but ignoring the part about the $b(v_1, \ldots, v_{m'})$ component; so only one rule is added in this case.

▶ **Lemma 3.2** (Correctness of saturation). *For $\mathcal{A}$ and $\mathcal{B}$ be as above, $\mathcal{L}(\mathcal{B}, P) = \mathrm{Pre}^*(\mathcal{L}(\mathcal{A}, P))$.*

The correctness proof, even though short, is presented in App. A of the technical report [11]. The right-in-left inclusion is by straightforward induction on the number of rewrite steps to reach $\mathcal{L}(\mathcal{A}, P)$. The left-in-right inclusion is more subtle, but with an appropriate invariant of the saturation process it also follows by a direct inspection.

3.2 Complexity

The reachability problem for AOTPSs can be solved using the saturation procedure from Theorem 3.1. For an initial configuration $(p_0, t_0) \in \mathcal{C}$ and an automaton $\mathcal{A}$ recognizing a regular set of target configurations $\mathcal{L}(\mathcal{A}, P)$, we construct $\mathcal{B}$ as in the previous section, and then test $(p_0, t_0) \in \mathcal{L}(\mathcal{B}, P)$. In this section we will analyze the complexity of this procedure in several relevant cases. All lower-bounds follow from the reductions presented in Sec. 4.

Let $m > 1$ be the maximal rank of any symbol in Σ. Using the notation from the previous subsection, we have that $|Q'_{n+1}| \leqslant |Q| + |\mathcal{R}|$, $|Q'_n| \leqslant |Q'_{n+1}| + |\mathcal{R}|$, and for every $k \in \{1, \ldots, n-1\}$, $|Q'_k| \leqslant |Q'_{k+1}| + |\mathcal{R}| \cdot 2^{(m-1)\cdot|Q'_{k+1}|} \leqslant O\left(|\mathcal{R}| \cdot 2^{(m-1)\cdot|Q'_{k+1}|}\right)$, and thus $|Q'| \leqslant \mathsf{exp}_{n-1}(O((m-1)\cdot(|Q| + |\mathcal{R}|)))$, where $\mathsf{exp}_0(x) = x$ and, for $i \geqslant 0$, $\mathsf{exp}_{i+1}(x) = 2^{\mathsf{exp}_i(x)}$. The size of the transition relation is at most one exponential more than the number of states, thus $|\Delta'| \leqslant \mathsf{exp}_n(O((m-1)\cdot(|Q| + |\mathcal{R}|)))$. This implies:

[4] This assumption is w.l.o.g. since one can always add shallow rules to reorder subtrees and put them in the required form.

▶ **Theorem 3.3.** *Reachability in order-n AOTPSs is* n-EXPTIMEc.

We identify four subclasses of AOTPSs, for which the reachability problem is of progressively decreasing complexity. First, we can save one exponential if we consider control-state reachability for the class of *non-deterministic, flat* AOTPSs. A system is *non-deterministic* when for every rule $p, l \to S, r$, the set S is a singleton. A system is *flat* when its rules $p, l \to S, r$ are flat (defined on page 167). *Control-state reachability* of a given set of locations $T \subseteq P$ means that the language of final configurations is $T \times \mathcal{T}(\Sigma)$. A proof of the theorem below is presented in App. B of the technical report [11].

▶ **Theorem 3.4.** *Control-state reachability in order-n non-deterministic flat AOTPSs is* $(n-1)$-EXPTIMEc, *where* $n \geqslant 2$.

Second, we consider the class of *linear* non-deterministic systems. Suppose that we consider *non-deterministic reachability*, i.e., that $\mathcal{A}$ is non-deterministic. When $\mathcal{S}$ is linear, i.e., variables in the r.h.s. of rules in $\mathcal{R}$ appear exactly once, then all P_i^t's and S_i^t's in (1) are singletons, and thus $\mathcal{B}$ is also non-deterministic. Consequently, the only states from $Q'_i \backslash Q'_{i+1}$ that are used by rewriting rules have the form $(g, \{p_{k+1}\}, \ldots, \{p_m\})$ for $p_{k+1}, \ldots, p_m \in Q'_{i+1}$. Therefore, there are at most $O((|Q| + |\mathcal{R}|)^{(m-1)^n})$ states and $O(|\mathcal{R}| \cdot |Q'|^m)$ transitions, and $\mathcal{B}$ is thus doubly exponential in n.

▶ **Theorem 3.5.** *The non-deterministic reachability problem in linear non-deterministic AOTPSs is* 2-EXPTIMEc.

The next simplification is when the system is *shallow* in the sense that it does not have deep rules. In this case we do not need to add states recursively ($Q' := Q \cup Q_0$), and we thus avoid the multiple exponential blow-up. Similarly, when the system is *unary*, i.e., the maximal rank is $m = 1$, only polynomially many states are added.

▶ **Theorem 3.6.** *Reachability in shallow as well as in unary AOTPSs is* EXPTIMEc.

If moreover the system is non-deterministic, then we get PTIME complexity, provided the rank of the letters in the alphabet is bounded.

▶ **Theorem 3.7.** *Non-deterministic reachability in unary non-deterministic AOTPSs and in shallow non-deterministic AOTPSs of fixed rank is in* PTIME.

3.3 Expressiveness

In the next section we give a number of examples of systems that can be directly encoded in AOTPSs. Before that, we would like to underline that AOTPSs can themselves be encoded into collapsible pushdown systems. We formally formulate this equivalence in terms of Krivine machines with states, which are defined later in Sec. 4.3. The details of this reduction are presented in App. E of the technical report [11].

▶ **Theorem 3.8.** *Every AOTPS of order n can be encoded in a Krivine machine with states of the same level s.t. every rewriting step of the AOTPS corresponds to a number of reduction steps of the Krivine machine.*

Since parity games over the configuration graph of the Krivine machine with states are known to be decidable [25], this equivalence yields decidability of parity games over AOTPSs. However, in this paper we concentrate on reachability properties of AOTPSs, which are decidable thanks to our simple saturation algorithm from Sec. 3.1. No such saturation algorithm was previously known for the Krivine machine with states.

4 Applications

In this section, we give several examples of systems that can be encoded as AOTPSs. Ordinary alternating pushdown systems (and even prefix-rewrite systems) can be easily encoded as *unary* AOTPSs by viewing a word as a linear tree; the ordering condition is trivial since symbols have rank $\leqslant 1$. Moreover, *tree-pushdown systems* [15] can be seen as *shallow* AOTPSs. By Theorem 3.6, reachability is in EXPTIME for both classes, and, by Theorem 3.7, it reduces to PTIME for the non-alternating variant (for fixed maximal rank).

In the rest of the section, we show how to encode four more sophisticated classes of systems, namely *ordered multi-pushdown systems* (Sec. 4.1), *annotated higher-order pushdown systems* (Sec. 4.2), the *Krivine machine with states* (Sec. 4.3), and *ordered annotated multi-pushdown systems* (Sec. 4.4), and we show that reachability for these models (except the last one) can be decided with tight complexity bounds using our conceptually simple saturation procedure.

4.1 Ordered multi-pushdown systems

In an ordered multi-pushdown system there are n pushdowns. Symbols can be pushed on any pushdown, but only the first non-empty pushdown can be popped [6, 3, 2]. This is equivalent to saying that to pop a symbol from the k-th pushdown, the contents of the previous pushdowns $1, \dots, k-1$ should be discarded. Formally, an *alternating ordered multi-pushdown system* is a tuple $\mathcal{O} = \langle n, \Gamma, Q, \Delta \rangle$, where $n \in \mathbb{N}_{>0}$ is the *order* of the system (i.e., the number of pushdowns), Γ is a finite pushdown alphabet, Q is a finite set of control locations, and $\Delta \subseteq Q \times O_n \times 2^Q$ is a set of rules of the form (p, o, P) with $p \in Q$, $P \subseteq Q$, and o a pushdown operation in $O_n := \{\mathsf{push}_k(a), \mathsf{pop}_k(a) \mid 1 \leqslant k \leqslant n, a \in \Gamma\}$. We say that $\mathcal{O}$ is *non-deterministic* when P is a singleton for every rule. A multi-pushdown system induces an alternating transition system $\langle \mathcal{C}_\mathcal{O}, \to_\mathcal{O} \rangle$ where the set of configurations is $\mathcal{C}_\mathcal{O} = Q \times (\Gamma^*)^n$, and the transitions are defined as follows: for every $(p, \mathsf{push}_k(a), P) \in \Delta$ there exists a transition $(p, w_1, \dots, w_n) \to_\mathcal{O} P \times \{(w_1, \dots, a \cdot w_k, \dots, w_n)\}$, and for every $(p, \mathsf{pop}_k(a), P) \in \Delta$ there exists a transition $(p, w_1, \dots, a \cdot w_k, \dots, w_n) \to_\mathcal{O} P \times \{(\varepsilon, \dots, \varepsilon, w_k, \cdots, w_n)\}$. For $c \in \mathcal{C}_\mathcal{O}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{O}$ asks whether $c \in \mathrm{Pre}^*(T \times (\Gamma^*)^n)$.

Encoding. We show that an ordered multi-pushdown system can be simulated by an AOTPS. The idea is to encode the k-th pushdown as a linear tree of order k, and to encode a multi-pushdown as a tree of linear pushdowns. Let $\perp$ and $\bullet$ be two new symbols not in Γ, let $\Gamma_\perp = \Gamma \cup \{\perp\}$, and let $\Sigma = (\Gamma_\perp \times \{1, \dots, n\}) \cup \{\bullet\}$ be an ordered alphabet, where a symbol $(a, i) \in \Gamma_\perp \times \{i\}$ has order i, rank 1 if $a \in \Gamma$ and rank 0 if $a = \perp$. Moreover, $\bullet$ has rank n and order 1. For simplicity, we write a^i instead of (a, i). A multi-pushdown $w_1, \dots, w_n$, where each $w_j = a_{j,1} \dots a_{j,n_j}$ is encoded as the tree $\mathsf{enc}(w_1, \dots, w_n) := \bullet(a^1_{1,1}(a^1_{1,2}(\dots \perp^1)), \dots, a^n_{n,1}(a^n_{n,2}(\dots \perp^n)))$. For an ordered multi-pushdown system $\mathcal{O} = \langle n, \Gamma, Q, \Delta \rangle$ we define an equivalent AOTPS $\mathcal{S} = \langle n, \Sigma, Q, \mathcal{R} \rangle$ with Σ defined as above, and set of rules $\mathcal{R}$ defined as follows (we use the convention that variable x_k has order k): For every push rule $(p, \mathsf{push}_k(a), P) \in \Delta$, we have a rule $(p, \bullet(\mathsf{x}_1, \dots, \mathsf{x}_n) \to P, \bullet(\mathsf{x}_1, \dots, a^k(\mathsf{x}_k), \dots, \mathsf{x}_n)) \in \mathcal{R}$, and for every pop rule $(p, \mathsf{pop}_k(a), P) \in \Delta$, we have $(p, \bullet(\mathsf{x}_1, \dots, a^k(\mathsf{x}_k), \dots, \mathsf{x}_n) \to P, \bullet(\perp^1, \dots, \perp^{k-1}, \mathsf{x}_k, \mathsf{x}_{k+1}, \dots, \mathsf{x}_n)) \in \mathcal{R}$. Both kinds of rules above are linear, and the latter one satisfies the ordering condition since lower-order variables $\mathsf{x}_1, \dots, \mathsf{x}_{k-1}$ are discarded. It is easy to see that $(p, w_1, \dots, w_n) \to^*_\mathcal{O} P \times \{(w'_1, \dots, w'_n)\}$ if, and only if, $(p, \mathsf{enc}(w_1, \dots, w_n)) \to^*_\mathcal{S} P \times \{\mathsf{enc}(w'_1, \dots, w'_n)\}$. Thus, the encoding preserves reachability properties. By Theorem 3.3, we obtain an n-EXPTIME upper-bound for reachability in alternating multi-pushdown systems of order n. Moreover, since $\mathcal{S}$ is linear, and

since $\mathcal{S}$ is non-deterministic when $\mathcal{O}$ is non-deterministic, by Theorem 3.5 we recover the optimal 2-EXPTIMEc complexity proved by [3] (cf. also [2]).

▶ **Theorem 4.1** ([3]). *Reachability in alternating ordered multi-pushdown systems is in* n-EXPTIME*, and* 2-EXPTIMEc *for the non-deterministic variant.*

Reachability for the alternating version of the model (in n-EXPTIME) was not previously known.

4.2 Annotated higher-order pushdown systems

Let Γ be a finite pushdown alphabet. In the following, we fix an order $n \geqslant 1$, and we let $1 \leqslant k \leqslant n$ range over orders. For our purpose, it is convenient to expose the topmost pushdown at every order recursively.[5] We define Γ_k, the set of *annotated higher-order pushdowns (stacks) of order* k, simultaneously for all $k \in \{1, \dots, n\}$, as the least set containing the empty pushdown $\langle\rangle$, and, whenever $u_1 \in \Gamma_1, \dots, u_k \in \Gamma_k$, $v_j \in \Gamma_j$ for some $j \in \{1, \dots, n\}$, then $\langle a^{v_j}, u_1, \dots, u_k\rangle \in \Gamma_k$. Similarly, if we do not consider stack annotations v_j's, we obtain the set of *higher-order pushdowns of order* k. Operations on annotated pushdowns are as follows. The operation push_k^b pushes a symbol $b \in \Gamma$ on the top of the topmost order-1 stack and annotates it with the topmost order-k stack, push_k duplicates the topmost order-$(k-1)$ stack, pop_k removes the topmost order-$(k-1)$ stack, and $\mathsf{collapse}_k$ replaces the topmost order-k stack with the order-k stack annotating the topmost symbol:

$$\mathsf{push}_k^b(\langle a^u, u_1, \dots, u_n\rangle) = \langle b^{\langle a^u, u_1, \dots, u_k\rangle}, \langle a^u, u_1\rangle, u_2, \dots, u_n\rangle,$$
$$\mathsf{push}_k(\langle a^u, u_1, \dots, u_n\rangle) = \langle a^u, u_1, \dots, u_{k-1}, \langle a^u, u_1, \dots, u_k\rangle, u_{k+1}, \dots, u_n\rangle,$$
$$\mathsf{pop}_k(\langle a^u, v_1, \dots, v_{k-1}, \langle b^v, u_1, \dots, u_k\rangle, u_{k+1}, \dots, u_n\rangle) = \langle b^v, u_1, \dots, u_n\rangle,$$
$$\mathsf{collapse}_k(\langle a^{\langle b^v, v_1, \dots, v_k\rangle}, u_1, \dots, u_n\rangle) = \langle b^v, v_1, \dots, v_k, u_{k+1}, \dots, u_n\rangle.$$

Let $O_n = \bigcup_{k=1}^n \{\mathsf{push}_k^b, \mathsf{push}_k, \mathsf{pop}_k, \mathsf{collapse}_k \mid b \in \Gamma\}$ be the set of stack operations. Similarly, one can define operations push^b and pop_k on stacks without annotations (but not $\mathsf{collapse}_k$, or push_k^b). An *alternating order-n annotated pushdown system* is a tuple $\mathcal{P} = \langle n, \Gamma, Q, \Delta\rangle$, where Γ is a finite stack alphabet, Q is a finite set of control locations, and $\Delta \subseteq Q \times \Gamma \times O_n \times 2^Q$ is a set of rules. An *alternating order-n pushdown system* (i.e., without annotations) is as $\mathcal{P}$ above, except that we consider non-annotated stack and operations on non-annotated stacks. An annotated pushdown system induces a transition system $\langle \mathcal{C}_\mathcal{P}, \to_\mathcal{P}\rangle$, where $\mathcal{C}_\mathcal{P} = Q \times \Gamma_n$, and the transition relation is defined as $(p, w) \to_\mathcal{P} P \times \{w'\}$ whenever $(p, a, o, P) \in \Delta$ with $w = \langle a^u, \cdots\rangle$ and $w' = o(w)$. Thus, a rule (p, a, o, P) first checks that the topmost stack symbol is a, and then applies the transformation provided by the stack operation o to the current stack (which may, or may not, change the topmost stack symbol a). Given $c \in \mathcal{C}_\mathcal{P}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{P}$ asks whether $c \in \mathrm{Pre}^*(T \times \Gamma_n)$.

Encoding. We represent annotated pushdowns as trees. Let Σ be the ordered alphabet containing, for each $k \in \{1, \dots, n\}$, an end-of-stack symbol $\bot^k \in \Sigma$ of rank 0 and order k. Moreover, for each $a \in \Gamma$ and order $k \in \{1, \dots, n\}$, there is a symbol $\langle a, k\rangle \in \Sigma$ of order k and rank $k+1$ representing the root of a tree encoding a stack of order k. An order-k stack is encoded as a tree recursively by $\mathsf{enc}_k(\langle\rangle) = \bot^k$ and $\mathsf{enc}_k(\langle a^u, u_1, \dots, u_k\rangle) = \langle a, k\rangle(\mathsf{enc}_i(u), \mathsf{enc}_1(u_1), \dots, \mathsf{enc}_k(u_k))$, where i is the order of u. Let $\mathcal{P} = \langle n, \Gamma, Q, \Delta\rangle$ be an

[5] Our definition is equivalent to [7].

annotated pushdown system. We define an equivalent AOTPS $\mathcal{S} = \langle n, \Sigma, Q, \mathcal{R}\rangle$, where Σ is as defined above, and $\mathcal{R}$ contains a rule $p, l \to P, r$ for each rule in $(p, a, o, P) \in \Delta$ and orders m, m_1, where $l \to r$ is as follows (cf. also Fig. 1 in the appendix of the technical report [11] for a pictorial representation). We use the convention that a variable subscripted by i has order i, and we write $\mathsf{x}_{i..j}$ for $(\mathsf{x}_i, \dots, \mathsf{x}_j)$, and similarly for $\mathsf{z}_{i..j}$:

$$\begin{aligned}
&\langle a, n\rangle(\mathsf{y}_m, \mathsf{x}_{1..n}) \to \langle b, n\rangle(\langle a, k\rangle(\mathsf{y}_m, \mathsf{x}_{1..k}), \langle a, 1\rangle(\mathsf{y}_m, \mathsf{x}_1), \mathsf{x}_{2..n}) && \text{if } o = \mathsf{push}^b_k,\\
&\langle a, n\rangle(\mathsf{y}_m, \mathsf{x}_{1..n}) \to \langle a, n\rangle(\mathsf{y}_m, \mathsf{x}_{1..k-1}, \langle a, k\rangle(\mathsf{y}_m, \mathsf{x}_{1..k}), \mathsf{x}_{k+1..n}) && \text{if } o = \mathsf{push}_k,\\
&\langle a, n\rangle(\mathsf{z}'_{m_1}, \mathsf{z}_{1..k-1}, \langle b, k\rangle(\mathsf{y}_m, \mathsf{x}_{1..k}), \mathsf{x}_{k+1..n}) \to \langle b, n\rangle(\mathsf{y}_m, \mathsf{x}_{1..n}) && \text{if } o = \mathsf{pop}_k,\\
&\langle a, n\rangle(\langle b, k\rangle(\mathsf{y}_m, \mathsf{x}_{1..k}), \mathsf{z}_{1..k}, \mathsf{x}_{k+1..n}) \to \langle b, n\rangle(\mathsf{y}_m, \mathsf{x}_{1..n}) && \text{if } o = \mathsf{collapse}_k.
\end{aligned}$$

The last two rules satisfy the ordering condition of AOTPSs since only higher-order variables $\mathsf{x}_{k+1}, \dots, \mathsf{x}_n$ are not discarded. It is easy to see that $(p, w) \to^*_{\mathcal{P}} P \times \{w'\}$ if, and only if, $(p, \mathsf{enc}_n(w)) \to^*_{\mathcal{S}} P \times \{\mathsf{enc}_n(w')\}$. Consequently, the encoding preserves reachability properties. Since an annotated pushdown system of order n is simulated by a flat AOTPS of the same order, the following complexity result is an immediate consequence of Theorems 3.3 and 3.4.

▶ **Theorem 4.2** ([7]). *Reachability in alternating annotated pushdown systems of order n and in non-deterministic annotated pushdown systems of order $n+1$ is* n-EXPTIMEc.

4.3 Krivine machine with states

We show that the Krivine machine evaluating simply-typed λY-terms can be encoded as an AOTPS. Essentially, this encoding was already given in the presentation of the Krivine machine operating on λY-terms from [24], though not explicitly given as tree pushdowns. In this sense, this provides the first saturation algorithm for the Krivine machine, thus yielding an optimal reachability procedure. Moreover, in App. E of the technical report [11] we present also a converse reduction (as announced earlier in Theorem 3.8), thus showing that the two models are in fact equivalent.

A *type* is either the basic type 0 or $\alpha \to \beta$ for types α, β. The *level* of a type is $\mathsf{level}(0) = 0$ and $\mathsf{level}(\alpha \to \beta) = \max(\mathsf{level}(\alpha) + 1, \mathsf{level}(\beta))$. We abbreviate $\alpha \to \dots \to \alpha \to \beta$ as $\alpha^k \to \beta$. Let $\mathcal{V} = \{x_1^{\alpha_1}, x_2^{\alpha_2}, \dots\}$ be a countably infinite set of typed variables, and let Γ be a ranked alphabet. A *term* is either

- **(i)** a constant $a^{0^k \to 0} \in \Gamma$,
- **(ii)** a variable $x^\alpha \in \mathcal{V}$,
- **(iii)** an abstraction $(\lambda x^\alpha . M^\beta)^{\alpha \to \beta}$,
- **(iv)** an application $(M^{\alpha \to \beta} N^\alpha)^\beta$, or
- **(v)** a fixpoint $(Y M^{\alpha \to \alpha})^\alpha$.

We sometimes omit the type annotation from the superscript, in order to simplify the notation. For a given term M, its set of *free variables* is defined as usual. A term M is *closed* if it does not have any free variable. We denote by $\Lambda(M)$ be the set of *sub-terms* of M. An *environment* ρ is a finite type-preserving function assigning closures to variables, and a *closure* C^α is a pair consisting of a term of type α and an environment, as expressed by the following mutually recursive grammar: $\rho ::= \varnothing \mid \rho[x^\alpha \mapsto C^\alpha]$ and $C^\alpha ::= (M^\alpha, \rho)$. We say that a closure (M, ρ) is *valid* if ρ binds all variables which are free in M (and no others), and moreover $\rho(x^\alpha)$ is itself a valid closure for each free variable x^α in M. Sometimes, we need to restrict an environment ρ by discarding some bindings in order to turn a closure (M, ρ) into a valid one. Given a term M and an environment ρ, the *restriction* of ρ to M, denoted $\rho\big|_M$, is obtained by removing from ρ all bindings for variables which are not

free in M. In this way, if (M,ρ) is a closure where ρ assigns valid closures to at least all variables which are free in M, then $(M,\rho|_M)$ is a valid closure. In a closure (M,ρ), M is called the *skeleton*, and it determines the type and level of the closure. Let $Cl^\alpha(M)$ be the set of valid closures of type α with skeleton in $\Lambda(M)$. An *alternating Krivine machine*[6] *with states* of level $l \in \mathbb{N}_{>0}$ is a tuple $\mathcal{M} = \langle l, \Gamma, Q, K^0, \Delta\rangle$, where $\langle \Gamma, Q, \Delta\rangle$ is an alternating tree automaton (in which a constant $a^{0^k\to 0} \in \Gamma$ is seen as a letter a of rank k), and K^0 is a closed term of type 0 s.t. the level of any sub-term in $\Lambda(K^0)$ is at most l. In the following, let $\alpha = \alpha_1 \to \cdots \to \alpha_k \to 0$. The Krivine machine $\mathcal{M}$ induces a transition system $\langle \mathcal{C}_\mathcal{M}, \to_\mathcal{M}\rangle$, where in a configuration $(p, C^\alpha, C_1^{\alpha_1}, \dots, C_k^{\alpha_k}) \in \mathcal{C}_\mathcal{M}$, $p \in Q$, $C^\alpha \in Cl^\alpha(K^0)$ is the *head closure*, and $C_1^{\alpha_1} \in Cl^{\alpha_1}(K^0), \dots, C_k^{\alpha_k} \in Cl^{\alpha_k}(K^0)$ are the *argument closures*. The transition relation $\to_\mathcal{M}$ depends on the structure of the skeleton of the head closure. It is deterministic except when the head is a constant in Γ, in which case the transitions in Δ control how the state changes (cf. also Fig. 2 in the appendix of the technical report [11] for a pictorial representation):

$$\begin{aligned}
(p, (x^\alpha, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k}) &\to_\mathcal{M} \{(p, \rho(x^\alpha), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})\},\\
(p, (M^\alpha N^{\alpha_1}, \rho), C_2^{\alpha_2}, \dots, C_k^{\alpha_k}) &\to_\mathcal{M} \{(p, (M^\alpha, \rho|_{M^\alpha}), (N^{\alpha_1}, \rho|_{N^{\alpha_1}}), C_2^{\alpha_2}, \dots, C_k^{\alpha_k})\},\\
(p, (Y M^{\alpha\to\alpha}, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k}) &\to_\mathcal{M} \{(p, (M^{\alpha\to\alpha}, \rho), ((YM)^\alpha, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})\},\\
(p, (\lambda x^{\alpha_0}.M^\alpha, \rho), C_0^{\alpha_0}, \dots, C_k^{\alpha_k}) &\to_\mathcal{M} \{(p, (M^\alpha, \rho[x^{\alpha_0} \mapsto C_0^{\alpha_0}]), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})\},\\
(p, (a^{0^k\to 0}, \rho), C_1^0, \dots, C_k^0) &\to_\mathcal{M} (P_1 \times \{C_1^0\}) \cup \cdots \cup (P_k \times \{C_k^0\})\\
&\qquad \text{for every } p \xrightarrow{a} P_1 \cdots P_k \in \Delta.
\end{aligned}$$

We say that $\mathcal{M}$ is *non-deterministic* if $\langle \Gamma, Q, \Delta\rangle$ is non-deterministic and all letters in Γ have rank at most 1. Given $c \in \mathcal{C}_\mathcal{M}$ and $T \subseteq Q$, the *(control-state) reachability problem* for $\mathcal{M}$ asks whether $c \in \mathrm{Pre}^*(T \times (\bigcup_{\alpha=\alpha_1\to\cdots\to\alpha_k\to 0} Cl^\alpha(K^0) \times Cl^{\alpha_1}(K^0) \times \cdots \times Cl^{\alpha_k}(K^0)))$.

Encoding. Following [24], we encode valid closures and configurations of the Krivine machine as ranked trees. Fix a Krivine machine $\mathcal{M} = \langle l, \Gamma, Q, K^0, \Delta\rangle$ of level l. We assume a total order on all variables $\langle x_1^{\beta_1}, \dots, x_n^{\beta_n}\rangle$ appearing in K^0. For a type α, we define $\mathsf{ord}(\alpha) = l - \mathsf{level}(\alpha)$. We construct an AOTPS $\mathcal{S} = \langle l, \Sigma, Q', \mathcal{R}\rangle$ of order l as follows. The ordered alphabet is

$$\Sigma = \{N^\alpha \mid N^\alpha \in \Lambda(K^0) \wedge \mathsf{level}(\alpha) < l\} \cup \{[N^\alpha] \mid N^\alpha \in \Lambda(K^0)\} \cup \{\bot_i \mid i \in \{1, \dots, n\}\}.$$

Here, N^α is a symbol of $\mathsf{rank}(N^\alpha) = n$ and $\mathsf{ord}(N^\alpha) = \mathsf{ord}(\alpha)$. Moreover, if $\alpha = \alpha_1 \to \cdots \to \alpha_k \to 0$ for some $k \geqslant 0$, then $[N^\alpha]$ is a symbol of $\mathsf{rank}([N^\alpha]) = n + k$ and $\mathsf{ord}([N^\alpha]) = l$ (in fact, $\mathsf{ord}([N^\alpha])$ is irrelevant, as $[N^\alpha]$ is used only in the root). Finally, $\bot_i$ is a leaf of order i. The set of control locations is $Q' = Q \cup \bigcup_{(p\xrightarrow{a} P_1\cdots P_k)\in\Delta} \{(1, P_1), \dots, (k, P_k)\}$. A closure (N^α, ρ) is encoded recursively as $\mathsf{enc}(N^\alpha, \rho) = N^\alpha(t_1, \dots, t_n)$, where, for every $i \in \{1, \dots, n\}$,

- **(i)** if $x_i \in \mathsf{FV}(N^\alpha)$ then $t_i = \mathsf{enc}(\rho(x_i))$, and
- **(ii)** $t_i = \bot_{\mathsf{ord}(\beta_i)}$ otherwise (recall that β_i is the type of x_i).

A configuration $c = (p, (N^\alpha, \rho), C_1^{\alpha_1}, \dots, C_k^{\alpha_k})$ is encoded as the tree $\mathsf{enc}(c) = [N^\alpha](t_1, \dots, t_n, \mathsf{enc}(C_1^{\alpha_1}), \dots, \mathsf{enc}(C_k^{\alpha_k}))$, where the first n subtrees encode the closure (N^α, ρ), i.e., $\mathsf{enc}(N^\alpha, \rho) = N^\alpha(t_1, \dots, t_n)$. The encoding is extended point-wise to sets of

[6] Cf. also [22] for a definition of the Krivine machine in a different context.

configurations. Notice that K^0 uses only variables of level at most $l-1$ (the subterm $\lambda x^\alpha.N$ introducing x^α is of level higher by one), so all skeletons in an environment are of order at most $l-1$. Similarly, skeletons in argument closures are of level at most $l-1$; only the head closure may have a skeleton of level l. Thus we do not need symbols N^α for $\mathsf{level}(\alpha) = l$.

Below, we assume that $\alpha = \alpha_1 \to \cdots \to \alpha_k \to 0$, that variable y_j has order $\mathsf{ord}(\alpha_j)$ for every $j \in \{0,\dots,k\}$, and that variables x_i and z_i have order $\mathsf{ord}(\beta_i)$ for every $i \in \{1,\dots,n\}$. Notice that $\mathsf{ord}(\alpha) < \mathsf{ord}(\alpha_1), \dots, \mathsf{ord}(\alpha_k)$. Moreover, we write $\mathbf{x} = \langle \mathsf{x}_1,\dots,\mathsf{x}_n\rangle$, $\mathbf{z} = \langle \mathsf{z}_1,\dots,\mathsf{z}_n\rangle$, and $\mathbf{y} = \langle \mathsf{y}_1,\dots,\mathsf{y}_k\rangle$. Finally, by $\mathbf{x}\big|_M$ we mean the tuple which is the same as $\mathbf{x}$, except that positions corresponding to variables not free in M are replaced by the symbol $\bot_{\mathsf{ord}(\beta_i)}$. $\mathcal{R}$ contains the following rules:

$$
\begin{aligned}
&p, [x_i^\alpha](\mathsf{z}_1,\dots,\mathsf{z}_{i-1}, M^\alpha(\mathbf{x}), \mathsf{z}_{i+1},\dots,\mathsf{z}_n,\mathbf{y}) \to \{p\}, [M^\alpha](\mathbf{x},\mathbf{y}),\\
&p, [M^\alpha N^{\alpha_1}](\mathbf{x},\mathsf{y}_2,\dots,\mathsf{y}_k) \to \{p\}, [M^\alpha](\mathbf{x}\big|_{M^\alpha}, N^{\alpha_1}(\mathbf{x}\big|_{N^{\alpha_1}}),\mathsf{y}_2,\dots,\mathsf{y}_k),\\
&p, [YM^{\alpha\to\alpha}](\mathbf{x},\mathbf{y}) \to \{p\}, [M^{\alpha\to\alpha}](\mathbf{x}, YM^{\alpha\to\alpha}(\mathbf{x}),\mathbf{y}),\\
&p, [\lambda x_i^{\alpha_0}.M^\alpha](\mathbf{x},\mathsf{y}_0,\mathbf{y}) \to \{p\}, [M^\alpha](\mathsf{x}_1,\dots,\mathsf{x}_{i-1},y_0,\mathsf{x}_{i+1},\dots,\mathsf{x}_n,\mathbf{y}),\\
&p, [a^{0^k\to 0}](\mathbf{x},\mathbf{y}) \to \{(1,P_1),\dots,(k,P_k)\}, [a^{0^k\to 0}](\mathbf{x},\mathbf{y}) \qquad \forall (p \xrightarrow{a} P_1\cdots P_k) \in \Delta,\\
&(i,P_i), [a^{0^k\to 0}](\mathbf{z},\mathsf{y}_1,\dots,\mathsf{y}_{i-1}, M_i^0(\mathbf{x}),\mathsf{y}_{i+1},\dots,\mathsf{y}_k) \to P_i, [M_i^0](\mathbf{x}).
\end{aligned}
$$

The first rule satisfies the ordering condition since the shared variables y_i are of order strictly higher than $\mathsf{ord}(M^\alpha)$. A direct inspection of the rules shows that, for a configuration c and a set of configurations D, we have $c \to^*_{\mathcal{M}} D$ if, and only if, $\mathsf{enc}(c) \to^*_{\mathcal{S}} \mathsf{enc}(D)$. Therefore, the encoding preserves reachability properties. Since a Krivine machine of level n is simulated by a flat AOTPS of order n, the following is an immediate consequence of Theorems 3.3 and 3.4.

▶ **Theorem 4.3** ([1]). *Reachability in alternating Krivine machines with states of level n and in non-deterministic Krivine machines with states of level $n+1$ is* n-EXPTIMEc.

4.4 Ordered annotated multi-pushdown systems

Ordered annotated multi-pushdown systems are the common generalization of ordered multi-pushdown systems and annotated pushdown systems [16]. Such a system is comprised of $m > 0$ annotated higher-order pushdowns arranged from left to right, where each pushdown is of order $n > 0$. While push operations are unrestricted, pop and collapse operations implicitly destroy all pushdowns to the left of the pushdown being manipulated, in the spirit of [6, 3, 2]. [16] has shown that reachability in this model can be decided in mn-fold exponential time, by using a saturation-based construction leveraging on the previous analysis for the first-order case [6, 3, 2]. In App. F of the technical report [11], we provide a simple encoding of an annotated multi-pushdown system with parameters (m,n) into an AOTPS of order mn. It is essentially obtained by taking together our previous encodings of ordered (cf. Sec. 4.1) and annotated systems (cf. Sec. 4.2). As a consequence of this encoding, by using the fact that an AOTPS of order mn can be encoded by a Krivine machine of the same level (by Theorem. 3.8), and by recalling the known fact that the latter can be encoded by a 1-stack annotated multi-pushdown system of order mn [26], we deduce that the concurrent behavior of an ordered m-stack annotated multi-pushdown system of order n can be *sequentialized* into a 1-stack annotated pushdown system of order mn (thus at the expense of an increase in order). The following complexity result is a direct consequence of Theorem 3.3.

▶ **Theorem 4.4** ([16]). *Reachability in alternating ordered annotated multi-pushdown systems of parameters (m,n) is in* (mn)-EXPTIME.

We remark that our result is for alternating systems, while [16] considers non-deterministic systems and obtain $(\mathsf{m}(\mathsf{n}-1))$-EXPTIME complexity. It seems that their method can be extended to alternating systems, and then the complexity becomes (mn)-EXPTIME as well.

5 Safety

The notion of safety has been made explicit by Knapik, Niwiński, and Urzyczyn [20] who identified the class of *safe recursive schemes*. They have shown that this class defines the same set of infinite trees as higher-order pushdown systems, i.e., the systems from Sec. 4.2 but without annotations. Blum and Ong [4] have extended the notion of safety to the simply-typed λ-calculus in a clear way. Then [26] adapted it to λY-calculus, and have shown that safe λY-terms correspond to higher-order pushdown automata without annotation.

There is a simple notion of safety for AOTPSs that actually corresponds to safety for pushdown systems and terms. We say that a $(\Sigma \cup \mathcal{V})$-tree is *safe* when looking from the root to the leafs the order does never increase. Formally, a tree u is *safe* if every subtree t thereof has order $\mathsf{ord}(t) \leqslant \mathsf{ord}(u)$ and it is itself safe. A rewrite rule $l \to r$ is *safe* if both l and r are safe. We say that $\mathcal{S}$ is *safe* if all its rules are safe.

As a first example, let us look at the encoding of annotated higher-order pushdown systems from Sec. 4.2. If we drop annotation then higher-order pushdowns are represented by safe trees, and all the rules are safe in the sense above. The case of Krivine machines is more difficult to explain, because it would need the definition of safety from [26]. In particular, one would have to partition variables into *lambda-variables* and *Y-variables*, which we avoid in the current presentation for simplicity. In the full version of the paper we will show that safe terms are encoded by safe trees, and that all the rules of the encoding of the Krivine machine preserve safety. Finally, we remark that the translation from AOTPSs to the Krivine machine with states previously announced in Theorem 3.8 can be adapted to produce a safe Krivine machine with states from a safe AOTPS.

6 Conclusions

We have introduced a novel extension of pushdown automata which is able to capture several sophisticated models thanks to a simple ordering condition on the tree-pushdown. While ordered tree-pushdown systems are not more expressive than annotated higher-order pushdown systems, or than Krivine machines, they offer some conceptual advantages. Compared to Krivine machines, they have states, and typing is replaced by a lighter mechanism of ordering; for example, the translation from our model back to the Krivine machine is much more cumbersome. Compared to annotated pushdown automata, the tree-pushdown is more versatile than a higher-order stack; for example, one can compare the encoding of the Krivine machine into our model to its encoding to annotated pushdown automata. We hope that ordered tree-pushdown systems will help to establish more connections with other models, as we have done in this paper with multi-pushdown systems.

There exist restrictions of multi-pushdown systems that we do not cover in this paper. Reachability games are decidable for phase-bounded multi-pushdown systems [27]. We can encode the phase-bounded restriction directly in our tree-pushdown systems, but we do not know how to deal with the scope-bounded restriction. Encoding the scope-bounded restriction would give an algorithm for reachability games over such systems, but we do not know if the problem is decidable.

Our general saturation algorithm can be used to verify reachability properties. We plan to extend it to the more general *parity properties*, in the spirit of [18]. We leave as future

work implementing our saturation algorithm, leveraging on subsumption techniques to keep the search space as small as possible.

Acknowledgments. We kindly acknowledge stimulating discussions with Irène Durand, Géraud Sénizergues, and Jean-Marc Talbot, and the anonymous reviewers for their helpful comments.

References

1 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Log. Methods Comput. Sci.*, 3(1):1–23, 2007.
2 M. Atig. Model-checking of ordered multi-pushdown automata. *Log. Methods Comput. Sci.*, 8(3), 09 2012.
3 M. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *Proc. of DLT'08*, pages 121–133. Springer, 2008.
4 W. Blum and C.-H. L. Ong. The safe lambda calculus. *Log. Methods Comput. Sci.*, 5(1), 2009.
5 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking and saturation method. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150, 1997.
6 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
7 C. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *Proc. of ICALP'12*, volume 7392 of *LNCS*, pages 165–176, 2012.
8 C. Broadbent, A. Carayol, M. Hague, and O. Serre. C-SHORe: A collapsible approach to higher-order verification. In *Proc. of ICFP '13*, pages 13–24. ACM, 2013.
9 C. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *In Proc. of CSL'13*, pages 129–148, 2013.
10 A. Carayol and M. Hague. Saturation algorithms for model-checking pushdown systems. In *Proc. of AFL'14*, volume 151 of *EPTCS*, pages 1–24, 5 2014.
11 L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. Ordered tree-pushdown systems. Technical report, University of Warsaw, October 2015. `http://arxiv.org/abs/1510.03278`.
12 A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *In Proc. of CONCUR'12*, pages 547–561, 2012.
13 J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *Proc. of CAV'01*, pages 324–336. Springer-Verlag, 2001.
14 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. of INFINITY'97*, volume 9, pages 27–37, 1997.
15 I. Guessarian. Pushdown tree automata. *Theor. Comp. Sys.*, 16:237–263, 1983.
16 Matthew H. Saturation of concurrent collapsible pushdown systems. In *Proc. of FSTTCS'13*, volume 24 of *LIPIcs*, pages 313–325, Dagstuhl, Germany, 2013.
17 M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proc. of LICS'08*, pages 452–461, 2008.
18 M. Hague and C.-H. L. Ong. A saturation method for the modal mu-calculus over pushdown systems. *Inform. and Comput.*, 209(5):799 – 821, May 2011.
19 A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. In *Proc. of MFCS'12*, pages 566–577. Springer, 2012.
20 T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Proc. of FOSSACS'02*, volume 2303 of *LNCS*, pages 205–222, 2002.

21 T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proc. of ICALP'05*, pages 1450–1461. Springer-Verlag, 2005.

22 J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher Order Symbol. Comput.*, 20:199–207, September 2007.

23 A. Maslov. Multilevel stack automata. *Probl. Peredachi Inf.*, 12(1):55–62, 1976.

24 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proc. of ICALP'11*, volume 6756 of *LNCS*, pages 162–173. Springer-Verlag, 2011.

25 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239(0):340–355, 2014.

26 S. Salvati and I. Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Math. Struct. in Comp. Science*, pages 1–47, 5 2015.

27 A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Proc. of CAV'10*, volume 6174 of *LNCS*, pages 615–628. Springer, 2010.

One-way Definability of Sweeping Transducers*

Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis

Université de Bordeaux, LaBRI & CNRS, France
{fbaschen,ogauwin,anca,gpuppis}@labri.fr

Abstract

Two-way finite-state transducers on words are strictly more expressive than one-way transducers. It has been shown recently how to decide if a two-way functional transducer has an equivalent one-way transducer, and the complexity of the algorithm is non-elementary. We propose an alternative and simpler characterization for sweeping functional transducers, namely, for transducers that can only reverse their head direction at the extremities of the input. Our algorithm works in 2ExpSpace and, in the positive case, produces an equivalent one-way transducer of doubly exponential size. We also show that the bound on the size of the transducer is tight, and that the one-way definability problem is undecidable for (sweeping) non-functional transducers.

1998 ACM Subject Classification F. Theory of Computation, F.4.3 Formal Languages

Keywords and phrases Regular word transductions, sweeping transducers, one-way definability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.178

1 Introduction

Regular word languages form the best understood class of languages. They enjoy several characterizations, in particular by different kinds of finite-state automata. For instance, two-way finite-state automata have the same expressive power as one-way automata. This result has been established independently by Rabin and Scott [9] and Shepherdson [10]. Besides automata, regular languages have logical and algebraic characterizations, namely through monadic second-order logic and congruences of finite index.

Transducers extend automata by producing outputs with each transition. A run generates an output word by concatenating the words produced by its transitions. A transducer thus defines a relation over words. It is called functional when this relation is a function. For finite-state transducers, expressiveness is different than for finite-state automata. As an example, two-way transducers are strictly more expressive than one-way transducers. For instance, the function that maps a word to its mirror image can be done by a back-and-forth pass over the input, but no one-way transducer can do it.

As seen above, we lose some robustness when going from automata to transducers. On the other hand, some of the classical characterizations of regular languages generalize well to transducers. An important result is the equivalence of functional two-way transducers and Ehrenfeucht-Courcelle's monadic-second order transductions [5] over words. Another characterization of two-way transducers was provided through a new model called streaming string transducers [1, 2], that process the input one-way and store the output in write-only

* This work was partially supported by the ExStream project (ANR-13-JS02-0010) and the Technische Universität München – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

registers. Finally, first-order transductions are known to be equivalent to aperiodic streaming transducers [7] and to aperiodic two-way transducers [4].

The question whether a functional two-way word transducer is equivalent to a one-way transducer has been solved recently in [6]. The algorithm proposed by [6] takes a two-way transducer $\mathcal{S}$ and builds a one-way transducer $\mathcal{T}$ that is "maximal" in the following sense: (1) all accepting runs of $\mathcal{T}$ produce correct outputs, and (2) all runs of $\mathcal{S}$ that can be performed one-way are realized by $\mathcal{T}$. As a consequence, the two-way transducer $\mathcal{S}$ has an equivalent one-way transducer if and only if the constructed one-way transducer $\mathcal{T}$ has the same domain as $\mathcal{S}$, which is a decidable problem. The problem is that the upper bound on both the decision procedure and the size of the constructed one-way transducer is non-elementary.

The main contribution of this paper is an elementary procedure for deciding whether a functional two-way word transducer is equivalent to a one-way transducer, for the particular class of sweeping transducers. While two-way transducers can reverse their head direction at any position of the input, sweeping transducers can only reverse it at the first and last position. Unsurprisingly, sweeping transducers are strictly less expressive than two-way transducers, and the following example shows the difference: on input $u_1\ a\ u_2\ a\ \dots\ a\ u_{n-1}\ a\ u_n$, where the words u_i contain no occurrence of a, the two-way transducer produces as output $u_n\ a\ u_{n-1}\ a\ \dots\ a\ u_2\ a\ u_1$ (we assume that the alphabet contains at least two letters).

Our decision procedure works in doubly exponential space and, when it succeeds, it produces an equivalent one-way transducer of doubly exponential size. We show that the bound on the size of the transducer is tight for any decision algorithm producing an equivalent one-way transducer from a sweeping transducer. This improves the PSPACE lower bound from [6]. The non-elementary procedure described in [6] relies on Rabin-Scott's construction for automata, and works by eliminating basic zigzags in runs. Our procedure is closer to the textbook approach (due to Shepherdson) and uses crossing sequences. This requires a decomposition of runs which is incomparable with the zigzag decomposition of [6]. Finally, we show that the one-way definability problem becomes undecidable for non-functional transducers.

Overview

Section 2 defines transducers and related concepts. Section 3 defines decomposition of runs and gives the construction of a one-way transducer based on such decompositions. In Section 4 we show that all one-way-definable runs admit a decomposition. Section 5 provides the lower bound and the undecidability result. A long version of the paper can be found on `http://www.labri.fr/perso/anca/Publications/fsttcs15.pdf`

2 Preliminaries

Transducers

A *two-way transducer* is a tuple $(\Sigma, \Delta, Q, I, F, \delta)$, where Σ (resp., Δ) is a finite input (resp., output) alphabet, Q is a finite set of states, I (resp., F) is a subset of Q representing the initial (resp., final) states, and $\delta \subseteq Q \times \Sigma \times \Delta^\star \times Q \times \{\mathsf{left}, \mathsf{right}\}$ is a finite set of transition rules describing, for each state and input symbol, the possible output string, target state, and direction of movement. We talk of a *one-way* transducer whenever $\delta \subseteq Q \times \Sigma \times \Delta^\star \times Q \times \{\mathsf{right}\}$. The *size* of a transducer is its number of states.

According to standard practice, the states of *one-way* automata and transducers are usually located between the letters of the input word $u = a_1 \dots a_n$. For this it is convenient

to introduce $n+1$ *positions* $0, 1, \ldots, n$ and think of each position $i > 0$ (resp., 0) as a placeholder between the i-th and the $i+1$-th symbols (resp., just before the first symbol a_1). Moreover, since here we deal with *two-way* devices, a single position can be visited several times along a run. Thus, to describe a run of a two-way transducer on input $u = a_1 \ldots a_n$, we will associate states with *locations*, namely, with pairs (x, y) where x is a position among $0, 1, \ldots, n$ and y is an integer representing the number of reversals performed up to a certain point – for short, we call this number y the *level* of the location.

A run is a sequence of locations, labelled by states and connected by edges, called *transitions*. The state at location $\ell = (x, y)$ of a run ρ is denoted $\rho(\ell)$. The transitions must connect pairs of locations that are either at adjacent positions and on the same level, or at the same position and on adjacent levels. In addition, each transition is labelled with a pair a/v consisting of an input symbol a and an output v. There are four types of transitions:

$$(i-1, 2y+1) \xleftarrow{a/v} (i, 2y+1) \qquad (i-1, 2y) \xrightarrow{a/v} (i, 2y)$$

$$a/v \circlearrowright \begin{matrix}(i, 2y+2)\\(i, 2y+1)\end{matrix} \qquad \begin{matrix}(i-1, 2y+1)\\(i-1, 2y)\end{matrix} \circlearrowleft a/v$$

Note that the transitions between locations at even levels are all directed from left to right, while the transitions at odd levels are directed from right to left. More precisely, the upper left (resp., upper right) transition may occur in a run ρ on $u = a_1 \ldots a_n$ if $\big(\rho(i, 2y+1), a, v, \rho(i-1, 2y+1), \mathsf{left}\big)$ (resp., $\big(\rho(i-1, 2y), a, v, \rho(i, 2y), \mathsf{right}\big)$) is a valid transition rule of $\mathcal{T}$ and $a = a_i$. Similarly, the lower left (resp., lower right) transition may occur if $\big(\rho(i, 2y+1), a, v, \rho(i, 2y+2), \mathsf{right}\big)$ (resp., $\big(\rho(i-1, 2y), a, v, \rho(i-1, 2y+1), \mathsf{left}\big)$) is a valid transition rule of $\mathcal{T}$ and $a = a_i$. For technical reasons (namely, to enable distinguished transitions at the extremities of the input word), we will introduce the special fresh symbols $\triangleright$ and $\triangleleft$ and allow the lower left (resp., lower right) transition also when $i = 0$ and $a = \triangleright$ (resp., when $i = |u|$ and $a = \triangleleft$).

Given a sequence $x_1, \ldots, x_n$, a *factor* denotes any contiguous subsequence $x_i, \ldots, x_j$, for $1 \leq i \leq j \leq n$. A run on the input $u = a_1 \ldots a_n$ is said to be *successful* if it starts at the lower left location $(0, 0)$ with an initial state of $\mathcal{T}$ and ends at the upper right location $(|u|, y_{\mathsf{max}})$ in a final state of $\mathcal{T}$. The output produced by a run is the concatenation of the outputs of its transitions, and it is denoted by $\mathsf{out}(\rho)$. We denote by $\mathsf{dom}(\mathcal{T})$ the language of all words u that admit a successful run of $\mathcal{T}$. We order the locations along a run ρ by letting $\ell_1 < \ell_2$ if ℓ_2 is reachable from ℓ_1 following the transitions in ρ. Given two locations $\ell_1 < \ell_2$ of a run ρ, we denote by $\rho[\ell_1, \ell_2]$ the factor of the run that starts in ℓ_1 and ends in ℓ_2. Note that $\rho[\ell_1, \ell_2]$ is also a run, hence the notation $\mathsf{out}\big(\rho[\ell_1, \ell_2]\big)$ is consistent.

Further assumptions

We will mostly work with two-way transducers that are *sweeping*. This means that on every successful run, the head can change direction only at the extremities of the input. In other words, the lower right (resp., lower left) transition is possible only if $a = \triangleleft$ (resp., $a = \triangleright$).

A transducer $\mathcal{T}$ is *functional* if, for each input word u, all successful runs on u produce the same output. In this case $\mathcal{T}(u)$ denotes the unique output produced on input u.

Unless otherwise stated, we will assume that all transducers are sweeping and functional. Note that functionality is a decidable property, as stated below. The proof is similar to the decidability proof of equivalence of streaming string transducers [1] and reduces the problem to the reachability of a 1-counter automaton of exponential size.

▶ **Proposition 1.** *Functionality of two-way transducers can be decided in polynomial space. This problem is* PSPACE*-hard even for sweeping transducers.*

Without loss of generality, we can also assume that the successful runs of a functional transducer are *normalized*, namely, they never visit two locations with the same position, the same state and both either at an even level or at an odd level. Indeed, if this were not the case, say if a successful run ρ visits two locations $\ell_1 = (x, y_1)$ and $\ell_2 = (x, y_2)$ such that $\rho(\ell_1) = \rho(\ell_2)$ and y_1, y_2 are both even or both odd, then the output produced by ρ between ℓ_1 and ℓ_2 is either empty – in which case we could remove $\rho[\ell_1, \ell_2]$ and obtain an equivalent successful run – or is non-empty – in which case, by repeating $\rho[\ell_1, \ell_2]$, we could obtain successful runs that produces different outputs on the same input, thus contradicting the assumption that the transducer is functional.

Crossing sequences

Consider a run ρ of a transducer on input $u = a_1 \dots a_n$. For each position $x \in \{0, 1 \dots, n\}$, we are interested in the sequence of states labelling the locations at position x. Formally, we define the *crossing sequence of* ρ *at* x as the tuple $\rho|x = \big(\rho(x, y_0), \dots, \rho(x, y_h)\big)$, where $y_0 < \dots < y_h$ are exactly the levels of the locations of ρ of the form (x, y), with $y \in \mathbb{N}$ (if the transducer is sweeping, we simply have $y_i = i$). If the considered run ρ is successful, then the bottom and top locations at position x have even levels, and the outgoing transitions move rightward. In particular, every crossing sequence of a successful run has odd length. Moreover, if we assume that the successful run is normalized, then every crossing sequence has length at most $2|Q| - 1$. The *crossing number* of a run is the maximal length of a crossing sequence of that run. The *crossing number* of a transducer is the maximal crossing number of any of its normalized runs – note that this value is bounded by $2|Q| - 1$.

Intercepted factors

An *interval* of positions has the form $I = [x_1, x_2]$, with $x_1 < x_2$. We say that an interval $I = [x_1, x_2]$ *contains* (resp., *strongly contains*) another interval $I' = [x'_1, x'_2]$ if $x_1 \leq x'_1 \leq x'_2 \leq x_2$ (resp., $x_1 < x'_1 \leq x'_2 < x_2$). We say that a factor of a run ρ is *intercepted* by an interval $I = [x_1, x_2]$ if it is maximal among the factors of ρ that visit only positions in I and that never make a reversal (recall that reversals in sweeping transducers can only occur at the extremities of the input word).

It is easy to see that distinct factors intercepted by the same interval I visit disjoint sets of locations. This means that a factor intercepted by I can be uniquely identified by specifying a location ℓ in it, e.g., the first or the last one. Accordingly, we will denote by $\rho \mid I/\ell$ the factor intercepted by I that visits the location ℓ (if this factor does not exist, we simply let $\rho \mid I/\ell = \varepsilon$).

A *loop* of a run ρ is an interval $L = [x_1, x_2]$ such that the crossing sequences at positions x_1 and x_2 are equal, that is, $\rho|x_1 = \rho|x_2$. Loops can be used to pump parts of runs, as explained below.

Pumping

Given a loop $L = [x_1, x_2]$ of a run ρ on u and a number $m \in \mathbb{N}$, we can replicate m times the factor $u[x_1, x_2]$ of the input and simultaneously, on the run, we replicate m times the loop L. Formally, with $\beta_0, \dots, \beta_h$ denoting the factors intercepted by L, we define the run obtained by replicating L as a sequence of the form

$$\mathsf{pump}_L^m(\rho) \;=\; \underbrace{\alpha_0\,\boldsymbol{\beta_0^m}\,\gamma_0}_{\text{forward}}\;\underbrace{\gamma_1\,\boldsymbol{\beta_1^m}\,\alpha_1}_{\text{backward}}\;\underbrace{\alpha_2\,\boldsymbol{\beta_2^m}\,\gamma_2}_{\text{forward}}\;\cdots\;\underbrace{\alpha_{y_{\max}}\,\boldsymbol{\beta_h^m}\,\gamma_{y_{\max}}}_{\text{forward}} \tag{1}$$

where $h < 2|Q| - 1$ is the maximum level visited by ρ, and α_y (resp., β_y, γ_y) is the factor of ρ at level y that is intercepted by the interval $[0, x_1]$ (resp., $L = [x_1, x_2]$, $[x_2, |u|]$). Note that each $\alpha_y\,\beta_y\,\gamma_y$ (resp., $\gamma_y\,\beta_y\,\alpha_y$) is a maximal factor of the run ρ that is forward-oriented (resp., backward-oriented). We also define

$$\mathsf{pump}_L^m(u) \;=\; u[1, x_1] \cdot \big(\boldsymbol{u[x_1 + 1, x_2]}\big)^{\boldsymbol{m}} \cdot u[x_2 + 1, |u|]$$

and we observe that $\mathsf{pump}_L^m(\rho)$ is a valid run on $\mathsf{pump}_L^m(u)$. We also remark that the above definition of pumped run works only for sweeping transducers, as for arbitrary two-way transducers we would need to take into account the possible reversals within a loop L and combine the intercepted factors in a more complex way.

3 Decompositions of one-way definable runs

The problem we consider in this paper is the *one-way definability* of functional, sweeping transducers: given a transducer $\mathcal{S}$, we ask if there exists an equivalent one-way transducer $\mathcal{T}$, namely, one such that $\mathcal{T}(u) = \mathcal{S}(u)$ for all $u \in \Sigma^\star$. In the answer is "yes", then we also want to compute an equivalent one-way transducer. Of course, there are sweeping transducers $\mathcal{S}$, like $\mathcal{S}(u) = u \cdot u$, that are not equivalent to any one-way transducer (assuming that the alphabet Σ is not unary).

Before introducing some technical concepts, let us consider an example that highlights the main idea of the proof. Fix a regular language R (not containing the empty word) and consider the transduction that maps a word on the mirror of the rightmost maximal factor belonging to R. That is, $f(u\,v\,w) = \mathsf{mirror}(v)$ whenever (1) $v \in R$, (2) there is no $v' \in R$ such that v' is prefix of vw, and (3) w has no factor in R. It can be easily seen that f can be realized by a two-way transducer, but not by a one-way transducer. However, f *can* be realized by a one-way transducer for particular regular languages R, like the periodic language $R = (ab)^+$: we simply guess v and output $\mathsf{mirror}(v) \in (ba)^+$ from left to right, then we check w. This example shows that periodicities play an important role in deciding whether a given transduction can be realized by a one-way transducer.

We introduce in the following some notations and concepts that will help us to state a sufficient condition for the one-way definability of sweeping transducers. For simplicity, we fix for the remaining of the paper a functional, sweeping transducer $\mathcal{S}$ as input. We first introduce some constants: $h_\mathcal{S}$ is the maximum number of levels visited by the normalized runs of $\mathcal{S}$, $c_\mathcal{S}$ is the maximum number of symbols produced by a single transition of $\mathcal{S}$, and $e_\mathcal{S} = c_\mathcal{S} \cdot |Q|^{2|Q|}$, where Q is the state space of $\mathcal{S}$. The constant $e_\mathcal{S}$ will be used to bound the lengths or the periods of certain parts of the output produced by $\mathcal{S}$, and is related to the number of crossing sequences of $\mathcal{S}$ (see also Lemma 6 in Section 4).

A word v is said to have *period* p if $v \in w^*\,w'$ for some word w of length p and some prefix w' of w. For example, $v = abc\,abc\,ab$ has period $p = 3$. Similarly, we say that v is *almost periodic with bound* p if $v = w_0\,w_1\,w_2$ for some words w_0, w_2 of length at most p and some non-empty word w_1 of period at most p.

We will also need to identify sub-sequences of a run ρ of $\mathcal{S}$ that are induced by particular sets of locations. Recall that $\rho[\ell_1, \ell_2]$ denotes the factor of ρ delimited by two locations ℓ_1 and ℓ_2. Similarly, we denote by $\rho \mid Z$ the sub-sequence of ρ induced by a set Z of locations – note that Z does not need to be an interval and, even though $\rho \mid Z$ might not be a valid run of $\mathcal{S}$, we can still refer to the number of transitions and the size of the output.

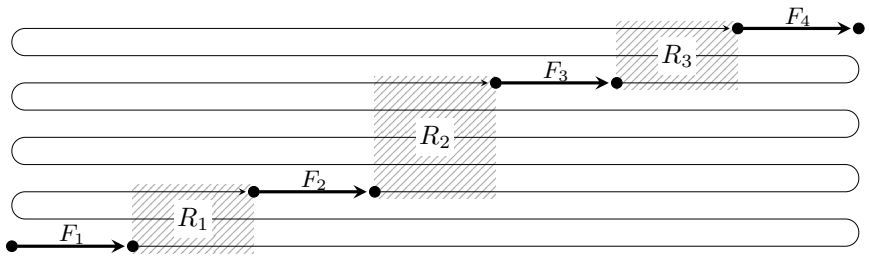
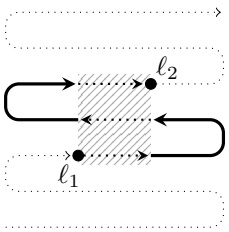


Figure 1 Decomposition of a run into floors and ramps.

▶ **Definition 2.** Let ρ be a run of $\mathcal{S}$ on u. We define two types of pairs of locations of ρ:

- A *floor* is a pair of locations (ℓ_1, ℓ_2) such that $\ell_1 \leq \ell_2$ are on the same *even* level.
- A *ramp* is a pair of locations (ℓ_1, ℓ_2), with $\ell_1 = (x_1, y_1)$ and $\ell_2 = (x_2, y_2)$, such that (i) $x_1 \leq x_2$, (ii) $y_1 < y_2$, (iii) both y_1 and y_2 are even, (iv) the output produced by $\rho[\ell_1, \ell_2]$ has length at most $(y_2 - y_1 + 1) \cdot e_{\mathcal{S}}$ or it is almost periodic with bound $2 \cdot e_{\mathcal{S}}$, and (v) the output produced by the sub-sequence $\rho \mid Z$, where $Z = [\ell_1, \ell_2] \setminus [x_1, x_2] \times [y_1, y_2]$, has length at most $2(y_2 - y_1) \cdot e_{\mathcal{S}}$.

Before discussing how the above definitions are used, we give some intuition. The simplest concept is that of floor, which is essentially a forward-oriented factor of a run. Ramps connect consecutive floors. An important constraint in the definition of a ramp (ℓ_1, ℓ_2) is that the output of $\rho[\ell_1, \ell_2]$ is bounded or almost periodic with small bound. We will see later how this constraint eases the production of the output of $\rho[\ell_1, \ell_2]$ by a one-way transducer. The last constraint on a ramp (ℓ_1, ℓ_2) bounds the length of the output produced by the sub-sequence $\rho \mid Z$, where $Z = [\ell_1, \ell_2] \setminus [x_1, x_2] \times [y_1, y_2]$. As shown by the figure to the right, this sub-sequence (represented by bold arrows) can be obtained from the factor $\rho[\ell_1, \ell_2]$ by removing the factors intercepted by $[x_1, x_2]$ (represented by the hatched area). The constraint is used for those parts of the run that are not covered by floors or ramps. In particular, it guarantees that the size of the output *above* each floor is bounded by $2h_{\mathcal{S}} \cdot e_{\mathcal{S}}$.



The general idea for turning $\mathcal{S}$ into an equivalent one-way transducer will be to guess (and check) a decomposition of the run of $\mathcal{S}$ into factors that are floors or ramps.

▶ **Definition 3.** A *decomposition* of a run ρ is a factorization into floors and ramps.

Figure 1 gives an example of a decomposition. Note that, thanks to Definition 2, the number of symbols produced outside the segments $F_1, F_2, \ldots$ and the rectangles $R_1, R_2, \ldots$ is small (indeed, bounded by $2h_{\mathcal{S}} \cdot e_{\mathcal{S}}$); so most of the output is produced inside these segments and rectangles. We can now state our main result:

▶ **Theorem 4.** *A sweeping functional transducer $\mathcal{S}$ is one-way definable if and only if every input word has some successful run of $\mathcal{S}$ that admits a decomposition.*
Moreover, we can construct from $\mathcal{S}$ a one-way transducer $\mathcal{T}$ that maps u to v whenever there is a successful run of $\mathcal{S}$ on u that outputs v and admits a decomposition. The construction of $\mathcal{T}$ takes doubly exponential time in $|\mathcal{S}|$.
In particular, $\mathcal{S}$ is one-way definable if and only if $\mathsf{dom}(\mathcal{T}) = \mathsf{dom}(\mathcal{S})$. The latter condition can be tested in polynomial space in $|\mathcal{S}|$ and $|\mathcal{T}|$, so in doubly exponential space in $|\mathcal{S}|$.

The first claim of the theorem gives the main characterization, namely, it shows that the existence of decompositions of successful runs, for all possible inputs in the domain of $\mathcal{S}$, is a *sufficient* and *necessary* condition for the transduction to be one-way definable. The second claim shows a property that is slightly more general than sufficiency: it allows to compute a one-way transducer $\mathcal{T}$ that is somehow the "best one-way approximation" of $\mathcal{S}$, in the sense that the transduction computed by $\mathcal{T}$ is always contained in the transduction computed by $\mathcal{S}$ and it is equal when $\mathcal{S}$ is one-way definable. The last claim deals with the *effectiveness* of the characterization, showing that one-way definability is decidable in 2ExpSpace. For the sake of presentation, we divide the proof of the theorem into two parts. The first part, given below, deals with the *sufficiency* and the *effectiveness* of the characterization (i.e. the second and third claims of the theorem). The second part, which is the most technical one and is deferred to Section 4, deals with the *necessary* part of the characterization.

Proof of Theorem 4 (sufficiency and effectiveness). We build from $\mathcal{S}$ a one-way transducer $\mathcal{T}$ that simulates all successful runs of $\mathcal{S}$ that admit a decomposition. Consider one such run ρ. We begin by observing that a decomposition of ρ can be described by a sequence of locations $\ell_0 < \ell_1 < \ldots < \ell_t$, where $\ell_0 = (0,0)$, $\ell_t = (x_{\mathsf{max}}, y_{\mathsf{max}})$, and, for all $0 \leq i < t$, (ℓ_i, ℓ_{i+1}) is a floor or a ramp. In particular, the one-way transducer $\mathcal{T}$ will guess the crossing sequences of ρ, together with a sequence of locations $(\ell_i)_{i \leq t}$, which are intended to represent a decomposition of ρ. Below, we show how to check that the guessed sequence of locations represents a valid decomposition, and how to produce the corresponding output.

Traversing the floors of the decomposition does not pose particular problems, as these are forward-oriented factors of the run ρ, which can be directly simulated by $\mathcal{T}$ without reversing the head. Of course, we need to store the bounded output on the levels above the floor, and check that the output on the levels below the floor matches some stored output words. The interesting case happens when $\mathcal{T}$ simulates a ramp (ℓ_i, ℓ_{i+1}), with $\ell_i = (x_i, y_i)$ and $\ell_{i+1} = (x_{i+1}, y_{i+1})$. First of all, it is easy for $\mathcal{T}$ to verify the first three conditions of the definition of ramp, namely, that $x_i \leq x_{i+1}$, $y_i < y_{i+1}$, and both y_i and y_{i+1} are even. Checking the remaining conditions is more difficult and requires storing some words for a total length that does not exceed $8h_{\mathcal{S}} \cdot e_{\mathcal{S}}$ – in particular, this explains the doubly exponential blowup of the state space of $\mathcal{T}$. More precisely, at the beginning of the computation, $\mathcal{T}$ guesses, for each ramp (ℓ_i, ℓ_{i+1}), with $\ell_i = (x_i, y_i)$ and $\ell_{i+1} = (x_{i+1}, y_{i+1})$:

- a word v_i that has length at most $(y_{i+1} - y_i + 1) \cdot e_{\mathcal{S}}$ or is almost periodic with bound $2 \cdot e_{\mathcal{S}}$ (in the latter case, in fact, the word is described by a prefix, a suffix, and a repeating pattern, each one of length at most $2 \cdot e_{\mathcal{S}}$),
- some words $\overleftarrow{v}_y$ and $\overrightarrow{v}_y$, that is, two words for each level $y \in \{y_i + 1, \ldots, y_{i+1}\}$, whose lengths sum up to at most $2h_{\mathcal{S}} \cdot e_{\mathcal{S}}$.

The idea is that each word v_i represents the output produced by the factor $\rho[\ell_i, \ell_{i+1}]$ (this output is bounded or is almost periodic, thanks to the fourth condition of the definition of ramp). Note that, by construction, there can be at most $h_{\mathcal{S}}$ ramps in the decomposition, and hence the sum of the lengths of the words used to represent the v_i's does not exceed $6 \cdot h_{\mathcal{S}} \cdot e_{\mathcal{S}}$. Similarly, each word $\overleftarrow{v}_y$ (resp., $\overrightarrow{v}_y$) represents the output produced by the factor of the run ρ that is at level y and to the left of the position x_i (resp., right of x_{i+1}), where i is the unique index such that $y_i < y \leq y_{i+1}$ (resp., $y_i \leq y < y_{i+1}$). The total length of these words is at most $2h_{\mathcal{S}} \cdot e_{\mathcal{S}}$. Overall, the sum of the lengths of all the words guessed by $\mathcal{T}$ never exceeds $8h_{\mathcal{S}} \cdot e_{\mathcal{S}}$.

Using the words v_i, $\overleftarrow{v}_y$, $\overrightarrow{v}_y$ and some additional pointers, the transducer $\mathcal{T}$ can verify that the guessed ramps satisfy the required conditions and that the decomposition is thus valid. In the same way, the words v_i, $\overleftarrow{v}_y$, $\overrightarrow{v}_y$ can be used to produce the output $\mathsf{out}(\rho[\ell_i, \ell_{i+1}])$

associated with each ramp (ℓ_i, ℓ_{i+1}). For this, it is sufficient to visit the positions of the ramp (ℓ_i, ℓ_{i+1}) and, at the same time, fetch blocks of symbols of appropriate length in the word v_i, so as to eventually match the length of the desired output $\mathsf{out}(\rho[\ell_i, \ell_{i+1}])$ – note that this requires taking into account also the words $\overleftarrow{v}_y$ and $\overrightarrow{v}_y$.

We just described informally a one-way transducer $\mathcal{T}$ that simulates any run ρ of $\mathcal{S}$ that admits a decomposition. The size of $\mathcal{T}$ is doubly exponential in the size of $\mathcal{S}$ and this proves the second claim of the theorem.

Assuming that the existence of decompositions of successful runs is also a necessary condition for the one-way definability of $\mathcal{S}$ (this will be proved in the next section), we can easily derive from the above constructions a 2ExpSpace decision procedure for testing one-way definability. More precisely, given a sweeping transducer $\mathcal{S}$, one constructs $\mathcal{T}$ as above in doubly exponential time, and then tests whether $\mathsf{dom}(\mathcal{S}) \subseteq \mathsf{dom}(\mathcal{T})$. The latter problem can be seen as a containment problem between a two-way non-deterministic finite-state automaton $\mathcal{A}$ and a one-way non-deterministic finite-state automaton $\mathcal{B}$. Using standard constructions, one can turn $\mathcal{A}$ into an equivalent one-way non-deterministic finite-state automaton $\mathcal{A}'$ (which is exponential in $\mathcal{S}$), and finally decide the containment $\mathcal{A}' \subseteq \mathcal{B}$ in polynomial space in $\mathcal{A}'$ and $\mathcal{B}$, that is, in doubly exponential space in $\mathcal{S}$. ◀

4 Characterizing one-way definability

In this section we prove the harder direction of the first claim of Theorem 4: if a sweeping functional transducer is one-way definable, then every accepted input has a successful run that can be decomposed into floors and ramps.

We begin by identifying some phenomena that prevent a transducer to be one-way definable. A first example is the mirror transduction, where a large number of symbols need to be generated from right to left. Another example is the doubling transduction $\mathcal{S}(u) = u \cdot u$. Here, we have an *inversion*, namely large parts of the input must be generated before other large parts that are located to their left. We give a formal definition of inversions below.

Fix a successful run ρ of $\mathcal{S}$ and consider a loop $L = [x_1, x_2]$ of ρ. A location ℓ_1 of ρ is called *entry point of* L if ℓ_1 is the first location of the factor intercepted by L at level y, for some y. Similarly, a location ℓ_2 is called an *exit point of* L if ℓ_2 is the last location of the factor intercepted by L at level k, for some k. Note that ℓ_1 belongs to $\{x_1\} \times (2\mathbb{N}) \cup \{x_2\} \times (2\mathbb{N}+1)$, and ℓ_2 belongs to $\{x_2\} \times (2\mathbb{N}) \cup \{x_1\} \times (2\mathbb{N}+1)$. Finally, we say that an intercepted factor $\rho \mid I/\ell$ is *captured* by a loop L if I contains L (possibly, $I = L$) and the subfactor intercepted by L on the same level as ℓ, has non-empty output.

▶ **Definition 5.** An *inversion* of the run ρ is a pair of locations ℓ_1 and ℓ_2 for which there exist two loops $L_1 = (x_1, x_1')$ and $L_2 = (x_2, x_2')$ such that:

- ℓ_1 is an entry point of L_1 and ℓ_2 is an exit point of L_2,
- $\ell_1 < \ell_2$ and $x_1 \geq x_2$ (namely, ℓ_2 strictly follows ℓ_1 along the run, but the left endpoint of L_2 precedes the left endpoint of L_1),
- for both $i = 1$ and $i = 2$, the intercepted factor $\rho \mid L_i/\ell_i$ is captured by the loop L_i, but it is not captured by any other loop strongly contained in L_i.

We say that the above loops L_1 and L_2 are *witnessing* the inversion (ℓ_1, ℓ_2).

The left-hand side of Figure 2 gives an example of an inversion, where the entry point ℓ_1 of L_1 and the exit point ℓ_2 of are represented by white circles.

The first lemma (proved in the appendix) can be used to bound the lengths of the outputs produced by the factors $\rho \mid L_1/\ell_1$ and $\rho \mid L_2/\ell_2$, where ℓ_1, ℓ_2, L_1, L_2 are as in Definition 5:

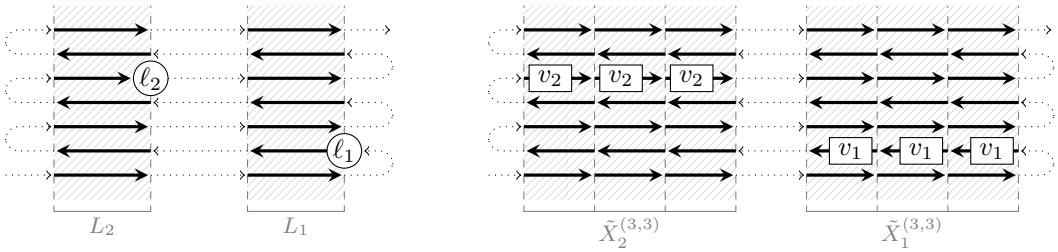


Figure 2 To the left: an entry point ℓ_1 of L_1 and an exit point ℓ_2 of L_2 forming an inversion. To the right: the run obtained by pumping the loops L_1 and L_2.

▶ **Lemma 6.** *If an intercepted factor $\rho \mid I/\ell$ is not captured by any loop L strongly contained in I, then the length of its output is at most $e_{\mathcal{S}}$.*
In particular, for every inversion (ℓ_1, ℓ_2) witnessed by some loops L_1 and L_2, we have $1 \leq \big|\mathsf{out}(\rho \mid L_i/\ell_i)\big| \leq e_{\mathcal{S}}$, for both $i = 1$ and $i = 2$.

The next proposition gives the crucial property for characterizing one-way definability, as it shows that the transducer $\mathcal{S}$ is one-way definable only if for every inversion (ℓ_1, ℓ_2), the output of $\rho[\ell_1, \ell_2]$ is periodic.

▶ **Proposition 7.** *Suppose that the sweeping transducer $\mathcal{S}$ is one-way definable. Then, for all inversions (ℓ_1, ℓ_2) of the run ρ witnessed by loops L_1, L_2 and for both $i = 1$ and $i = 2$, the output of $\rho[\ell_1, \ell_2]$ has period $\big|\mathsf{out}(\rho \mid L_i/\ell_i)\big|$, hence, in particular, at most $e_{\mathcal{S}}$.*

A key ingredient for the proof of the above proposition is Fine and Wilf's theorem [8]. In short, this theorem says that, whenever two periodic words w_1, w_2 share a sufficiently long factor, then they have the same periods. Below, we state a slightly stronger variant of Fine and Wilf's theorem, which contains an additional claim that shows how to align a common factor of the two words w_1, w_2 so as to form a third word containing a prefix of w_1 and a suffix of w_2. The additional claim will be exploited in the proof of Proposition 7 and Lemma 11.

▶ **Lemma 8** (Fine and Wilf). *If w_1 is a word with period p_1, w_2 is a word with period p_2, and w_1 and w_2 have a common factor of length at least $p_1 + p_2 - \gcd(p_1, p_2)$, then w_1 and w_2 have also period $\gcd(p_1, p_2)$. If in addition we have $w_1 = u_1\, w\, v_1$, $w_2 = u_2\, w\, v_2$, and $|w| \geq \gcd(p_1, p_2)$, then $w_3 = u_1\, w\, v_2$ has also period $\gcd(p_1, p_2)$.*

Proof of Proposition 7. Let $L_1 = (x_1, x_1')$, $L_2 = (x_2, x_2')$ be the loops witnessing the inversion (ℓ_1, ℓ_2). Note that the two loops L_1 and L_2 might not be ordered exactly as shown in Figure 2. In fact, two cases can arise: either $x_2 < x_2' \leq x_1 < x_1'$ (that is, L_1 and L_2 are disjoint and L_1 is to the right of L_2) or $x_2 \leq x_1 < x_2' \leq x_1'$ (that is, L_1 overlaps to the right with L_2).

We begin by pumping the loops L_1 and L_2 (see the right-hand side of Figure 2). Formally, for all positive numbers m_1, m_2, we define

$$\rho^{(m_1,m_2)} \;=\; \mathsf{pump}_{L_2}^{m_2}(\mathsf{pump}_{L_1}^{m_1}(\rho)) \qquad \text{and} \qquad u^{(m_1,m_2)} \;=\; \mathsf{pump}_{L_2}^{m_2}(\mathsf{pump}_{L_1}^{m_1}(u))\ .$$

We identify the positions of $u^{(m_1,m_2)}$ that mark the endpoints of the copies of the loops L_1 and L_2 in the pumped run $\rho^{(m_1,m_2)}$. Because L_2 precedes L_1 with respect to the ordering of positions, it is easier to define first the set of endpoints of the copies of L_2:

$$\tilde{X}_2^{(m_1,m_2)} \;=\; \{x_2 + i\Delta_2 \;\mid\; 0 \leq i \leq m_2\} \qquad\qquad \text{where} \quad \Delta_2 = x_2' - x_2\ .$$

The set of endpoints of the copies of L_1 is defined as

$$\tilde{X}_1^{(m_1,m_2)} \;=\; \{x_1 + i\Delta_1 + m_2\Delta_2 \;\mid\; 0 \le i \le m_1\} \qquad \text{where} \;\; \Delta_1 = x_1' - x_1 \;.$$

We then exploit the hypothesis that $\mathcal{S}$ is one-way definable and assume that the one-way transducer $\mathcal{T}$ is equivalent to $\mathcal{S}$. In particular, $\mathcal{T}$ produces the same output as $\mathcal{S}$ on every input $u^{(m_1,m_2)}$. Let $\sigma^{(m_1,m_2)}$ be a successful run of $\mathcal{T}$ on $u^{(m_1,m_2)}$. Since $\mathcal{T}$ has finitely many states, we can find a large enough number $h > 0$ and two positions $\tilde{x}_1 < \tilde{x}_1' \in \tilde{X}_1^{(h,h)}$ such that the crossing sequences of $\sigma^{(h,h)}$ at $\tilde{x}_1$ and $\tilde{x}_1'$ are the same. Similarly, we can find two positions $\tilde{x}_2 < \tilde{x}_2' \in \tilde{X}_2^{(h,h)}$ such that the crossing sequences of $\sigma^{(h,h)}$ at $\tilde{x}_2$ and $\tilde{x}_2'$ are the same. This means that $\tilde{L}_1 = (\tilde{x}_1, \tilde{x}_1')$ and $\tilde{L}_2 = (\tilde{x}_2, \tilde{x}_2')$ can be equally seen as loops of $\rho^{(h,h)}$ or as loops of $\sigma^{(h,h)}$. In particular, there are constants $k_1, k_2 > 0$, $0 \le h_1 < k_1$, and $0 \le h_2 < k_2$ such that, for all positive numbers m_1, m_2:

$$u^{(k_1\cdot m_1 + h_1,\, k_2\cdot m_2 + h_2)} \;=\; \mathsf{pump}_{\tilde{L}_2}^{m_2}(\mathsf{pump}_{\tilde{L}_1}^{m_1}(u^{(h,h)}))$$

and the above word has a successful run in $\mathcal{T}$ of the form $\mathsf{pump}_{\tilde{L}_2}^{m_2}(\mathsf{pump}_{\tilde{L}_1}^{m_1}(\sigma^{(h,h)}))$. Consider the outputs v_1 and v_2 produced by the intercepted factors $\rho \mid L_1/\ell_1$ and $\rho \mid L_2/\ell_2$, respectively. By Lemma 6, both v_1 and v_2 are non-empty. Moreover, by definition of pumped run, the output produced by $\rho^{(k_1\cdot m_1 + h_1,\, k_2\cdot m_2 + h_2)}$ contains $k_1 \cdot m_1 + h_1$ consecutive occurrences of v_1 followed by $k_2 \cdot m_2 + h_2$ consecutive occurrences of v_2 (see again the right-hand side Figure 2). Formally, we can write

$$\mathsf{out}(\rho^{(k_1\cdot m_1 + h_1,\, k_2\cdot m_2 + h_2)}) \;=\; v_0(m_1,m_2)\cdot \boldsymbol{v_1^{k_1\cdot m_1 + h_1}} \cdot v_3(m_1,m_2) \cdot \boldsymbol{v_2^{k_2\cdot m_2 + h_2}} \cdot v_4(m_1,m_2) \quad (2)$$

for some words $v_0(m_1,m_2)$, $v_3(m_1,m_2)$, and $v_4(m_1,m_2)$ that may depend on m_1 and m_2 (we highlighted in bold the repeated occurrences of v_1 and v_2 and we observe that v_1 precedes v_2).

In a similar way, because the same output is also produced by the the one-way transducer $\mathcal{T}$, i.e. by the run $\mathsf{pump}_{\tilde{L}_2}^{m_2}(\mathsf{pump}_{\tilde{L}_1}^{m_1}(\sigma^{(h,h)}))$, and because the loop L_2 precedes the loop L_1 according to the natural ordering of positions, we have

$$\mathsf{out}(\rho^{(k_1\cdot m_1 + h_1,\, k_2\cdot m_2 + h_2)}) \;=\; w_0 \cdot \boldsymbol{w_2^{m_2}} \cdot w_3 \cdot \boldsymbol{w_1^{m_1}} \cdot w_4 \quad (3)$$

where w_1 (resp., w_2) is the output produced by the unique factor of $\sigma^{(h,h)}$ intercepted by $\tilde{L}_1$ (resp., $\tilde{L}_2$), and w_0, w_3, w_4 are the remaining parts of the output. Note that, differently from the previous equation, here the first repetition is produced during the loop $\tilde{L}_2$ and the remaining parts w_0, w_3, w_4 do not depend on m_1, m_2. We now consider the factor

$$v(m_1,m_2) \;=\; \boldsymbol{v_1^{k_1\cdot m_1 + h_1}} \cdot v_3(m_1,m_2) \cdot \boldsymbol{v_2^{k_2\cdot m_2 + h_2}}$$

of the output produced by $\mathcal{S}$. The following claim shows that this factor is periodic, with a small period that only depends on $\mathcal{S}$ (in particular, it does not depend on any of the indices $h, k_1, k_2, h_1, h_2, m_1, m_2$).

▶ **Claim.** For all numbers $m_1, m_2 > 0$, the word $v(m_1,m_2) = \boldsymbol{v_1^{k_1\cdot m_1 + h_1}} \cdot v_3(m_1,m_2) \cdot \boldsymbol{v_2^{k_2\cdot m_2 + h_2}}$ is periodic with period $\gcd(|v_1|, |v_2|)$.

The idea for the proof of the above claim, detailed in the appendix, is to let m_1 and m_2 grow independently. We exploit Equations (2) and (3) to show that $\boldsymbol{v_1^{k_1\cdot m_1 + h_1}} \cdot v_3(m_1,m_2) \cdot \boldsymbol{v_2}$ has period $\gcd(|v_1|, |w_1|)$, and that $\boldsymbol{v_1} \cdot v_3(m_1,m_2) \cdot \boldsymbol{v_2^{k_2\cdot m_2 + h_2}}$ has period $\gcd(|v_2|, |w_2|)$. A last application of Fine and Wilf's theorem (Lemma 8) gives the desired periodicity.

Recall that we aim at proving the periodicity of the output $\mathsf{out}(\rho[\ell_1, \ell_2])$ of the *original* run ρ of $\mathcal{S}$ between the locations ℓ_1 and ℓ_2. The previous arguments, however, concern the outputs $v(m_1, m_2)$, which are produced by factors of the *pumped* runs $\rho^{(k_1 \cdot m_1 + h_1, k_2 \cdot m_2 + h_2)}$. By Equation (1) in Section 2, $\mathsf{out}(\rho[\ell_1, \ell_2])$ can be obtained from any $v(m_1, m_2)$ by deleting some occurrences of non-empty words produced by factors intercepted by L_1 or L_2. More precisely, the words that need to be deleted in $v(m_1, m_2)$ to obtain $\mathsf{out}(\rho[\ell_1, \ell_2])$ are non-empty and of the form $\mathsf{out}(\rho \mid L_i/\ell_i')$, for some $i \in \{1, 2\}$ and some location ℓ_i' such that $\ell_1 \leq \ell_i' \leq \ell_2$. Let us denote by $v_1', \ldots, v_m'$ these words. Note that, as m_1 and m_2 get larger, $v(m_1, m_2)$ contains arbitrarily long repetitions of each word v_i', and hence long factors of period $|v_i'|$, for $i = 1, \ldots, m$. Thus, by applying Lemma 8, we get that $v(m_1, m_2)$ has period $p = \gcd(|v_1|, |v_2|, |v_1'|, \ldots, |v_m'|)$.

Towards a conclusion, we know that $\mathsf{out}(\rho[\ell_1, \ell_2])$ is obtained from $v(m_1, m_2)$ by removing occurrences of the words $v_1', \ldots, v_m'$ whose lengths are multiple of the period p of $v(m_1, m_2)$. This implies that $\mathsf{out}(\rho[\ell_1, \ell_2])$ is also periodic with period p, which divides $\left|\mathsf{out}(\rho \mid L_i/\ell_i)\right|$ for both $i = 1$ and $i = 2$. ◀

Recall that the proof of the remaining part of Theorem 4 (necessity of the condition characterizing one-way definability) amounts at constructing a decomposition of the successful run ρ under the assumption that $\mathcal{S}$ is one-way definable. We begin to construct a decomposition of ρ by identifying some ramps in it. Intuitively, such ramps are obtained by considering the classes of a suitable equivalence relation:

▶ **Definition 9.** Let $\leftrightharpoons$ be the relation that pairs every two locations ℓ, ℓ' along the run ρ whenever there is an inversion (ℓ_1, ℓ_2) of ρ such that $\ell_1 \leq \ell, \ell' \leq \ell_2$, namely, whenever ℓ and ℓ' occur within the same inversion. Let $\leftrightharpoons^\star$ be the reflexive and transitive closure of $\leftrightharpoons$.

It is easy to see that every equivalence class of $\leftrightharpoons^\star$ is a convex subset with respect to the natural ordering of locations of ρ. The following lemma shows that every *non-singleton* equivalence class of $\leftrightharpoons^\star$ is a union of a series of inversions that are two-by-two overlapping.

▶ **Lemma 10.** *If two locations $\ell \leq \ell'$ of ρ belong to the same* non-singleton *equivalence class of $\leftrightharpoons^\star$, then there is a sequence of locations $\ell_1 \leq \ell_3 \leq \ell_4 \leq \ldots \leq \ell_{n-3} \leq \ell_{n-2} \leq \ell_n$, for some even number $n \geq 4$, such that*

- *$\ell_1 \leq \ell \leq \ell_4$ and $\ell_{n-3} \leq \ell' \leq \ell_n$,*
- *(ℓ_1, ℓ_4), (ℓ_3, ℓ_6), (ℓ_5, ℓ_8), $\ldots$, (ℓ_{n-5}, ℓ_{n-2}), (ℓ_{n-3}, ℓ_n) are inversions.*

The next result uses Lemma 6, Proposition 7, and Lemma 10 to show that the output produced inside a $\leftrightharpoons^\star$-equivalence class is also periodic with small period, provided that $\mathcal{S}$ is one-way definable.

▶ **Lemma 11.** *If $\mathcal{S}$ is one-way definable and $\ell \leq \ell'$ are two locations of the run ρ such that $\ell \leftrightharpoons^\star \ell'$, then the output $\mathsf{out}(\rho[\ell, \ell'])$ produced between ℓ and ℓ' has period at most $e_{\mathcal{S}}$.*

Below, we introduce some "bounding boxes" of non-singleton $\leftrightharpoons^\star$-equivalence classes. Intuitively, these bounding boxes are the smallest possible rectangles that start and end at some even levels and that cover all the locations forming an inversion inside a non-singleton $\leftrightharpoons^\star$-equivalence class. Subsequently, in Lemma 13 we show that these bounding boxes can be given the status of ramps in a suitable decomposition of ρ.

▶ **Definition 12.** Let K be a non-singleton $\leftrightharpoons^\star$-equivalence class and let H be the subset of K that contains all the locations $\ell, \ell' \in K$ forming an inversion (ℓ, ℓ').
We define $[K]$ to be the pair of locations $\ell_1 = (x_1, y_1)$ and $\ell_2 = (x_2, y_2)$ such that

- x_1 (resp., x_2) is the position of the leftmost (resp., rightmost) location $\ell \in H$,
- y_1 (resp., y_2) is the highest (resp., lowest) even level such that $y_1 \leq y$ (resp., $y_2 \geq y$) for all locations $\ell = (x, y) \in H$.

▶ **Lemma 13.** *If K is a non-singleton $\leftrightharpoons^\star$-equivalence class, then $[K]$ is a ramp.*

For the sake of brevity, we call $\leftrightharpoons^\star$-*ramp* any ramp of the form $[K]$, where K is a non-singleton $\leftrightharpoons^\star$-equivalence class. The results obtained so far imply that every location of the run ρ covered by an inversion is also covered by a $\leftrightharpoons^\star$-ramp. To complete the decomposition of ρ, we need to consider the locations that are not strictly covered by $\leftrightharpoons^\star$-ramps, formally, the set $B = \{\ell \mid \nexists\, (\ell_1, \ell_2)\ \leftrightharpoons^\star\text{-ramp s.t. } \ell_1 < \ell < \ell_2\}$. We equip B with the natural ordering of locations induced by ρ. We now consider some maximal convex subset C of B. Note that the left/right endpoint of C coincides with the first/last location of the run ρ or with the right/left endpoint of some $\leftrightharpoons^\star$-ramp. Below, we show how to decompose the sub-run $\rho \mid C$ into a series of floors and ramps. After this, we will be able to get a full decomposition of ρ by interleaving the $\leftrightharpoons^\star$-ramps that we defined above with the floors and the ramps that decompose each sub-run $\rho \mid C$.

Let D_C be the set of locations $\ell = (x, y)$ of C such that there is some loop $L = [x, x']$, with $x' \geq x$, whose intercepted factor $\rho \mid L/\ell$ lies entirely inside C and produces non-empty output. We remark that the set D_C may be non-empty. To see this, one can imagine the existence of two consecutive $\leftrightharpoons^\star$-ramps (e.g. R_1 and R_2 in Figure 1) and a loop between them that produces non-empty output (e.g. the factor F_2). In a more general scenario, one can find several loops between two consecutive $\leftrightharpoons^\star$-ramps that span across different levels. We can observe however that all the locations in D_C are on even levels. Indeed, if this were not the case for some $\ell = (x, y) \in D_C$, then we could select a minimal loop $L = [x_1, x_2]$ such that $x_1 \leq x \leq x_2$ and $\mathsf{out}(\rho \mid L/\ell) \neq \varepsilon$. Since y is odd, $\ell_1 = (x_2, y)$ is an entry point of L and $\ell_2 = (x_1, y)$ is an exit point of L, and hence (ℓ_1, ℓ_2) is an inversion. Since $\ell_1 \leq \ell \leq \ell_2$ and all inversions are covered by $\leftrightharpoons^\star$-ramps, there is a $\leftrightharpoons^\star$-ramp (ℓ'_1, ℓ'_2) such that $\ell'_1 \leq \ell \leq \ell'_2$. However, as ℓ'_1 and ℓ'_2 are at even levels, ℓ must be different from these two locations, and this would contradict the definition of D_C. Using similar arguments, one can also show that the locations in D_C are arranged along a "rising diagonal", from lower left to upper right.

The above properties suggest that the locations in D_C identify some floors and ramps that form a decomposition of $\rho \mid C$. The following lemma shows that this is indeed the case, namely, that any two consecutive locations in D_C form a floor or a ramp.

▶ **Lemma 14.** *Let $\ell_1 = (x_1, y_1)$ and $\ell_2 = (x_2, y_2)$ be two consecutive locations of D_C. Then, $x_1 \leq x_2$ and $y_1 \leq y_2$ and the pair (ℓ_1, ℓ_2) is a floor or a ramp, depending on whether $y_1 = y_2$ or $y_1 < y_2$.*

We have just shown how to construct a decomposition of the entire run ρ, assuming that the sweeping transducer $\mathcal{S}$ is one-way definable. This completes the proof of the only-if direction of the first claim of Theorem 4.

5 Lower bound and undecidability

We show now that the doubly exponential blow-up in size stated by Theorem 4, cannot be avoided.

▶ **Proposition 15.** *There is a family $(f_n)_n$ of functions from $\{0,1\}^*$ to $\{0,1\}^*$ such that:*

- *f_n can be computed by a sweeping transducer of size quadratic in n,*
- *f_n can be computed by a one-way transducer,*
- *any one-way transducer computing f_n has at least $2^{2^{n-1}}$ states.*

We exhibit such a family of function by defining the domain of f_n to be the set of words of the form

$$a_0 \ \mathsf{bin}_0 \ a_1 \ \mathsf{bin}_1 \ a_2 \ \ldots \ a_{2^n-1} \ \mathsf{bin}_{2^n-1} \ a_{2^n}$$

where $a_i \in \{0,1\}$ for all $i \in \{0,\ldots,2^n\}$. On those words, we define f_n as follows:

$$f_n\big(a_0 \ \mathsf{bin}_0 \ a_1 \ \mathsf{bin}_1 \ a_2 \ \ldots \ a_{2^n-1} \ \mathsf{bin}_{2^n-1} \ a_{2^n}\big) \ = \ w \cdot w \quad \text{where } w \ = \ a_0\, a_1 \ \ldots \ a_{2^n} \ .$$

We conclude the section by showing that the one-way definability problem becomes undecidable for relations computed by sweeping *non-functional* transducers. Note that ε-transitions are needed in order to capture the class of one-way definable relations. On the other hand, for one-way definable functions, ε-transitions can be excluded.

▶ **Proposition 16.** *The problem of testing whether a sweeping* non-functional *transducer is one-way definable is undecidable.*

6 Conclusion

In this paper we proposed a new algorithm that decides whether a sweeping transducer is equivalent to a one-way transducer. Our decision algorithm works in doubly exponential space and produces one-way transducers of doubly exponential size. The latter bound is shown to be optimal. An open question is whether the decision problem has lower complexity if we do not build the one-way transducer.

The main open question is whether our algorithm can be extended to two-way functional transducers that are not necessarily sweeping. We conjecture that this is the case and that a similar characterization based on decompositions of runs into floors and ramps can be obtained. The main difficulty is that (de)pumping loops is more complicated because of permutations.

One-way definability is also a special case of the following open problem: given an integer k and a two-way transducer, decide if there is an equivalent k-crossing two-way transducer. Finally, note that the problem that we considered here becomes much simpler in the origin semantics of [3]: there, the output of a transducer also includes the origin of each symbol, i.e., the input position where the symbol was generated. In the origin semantics, the one-way definability problem is PSpace-complete, and an equivalent one-way transducer has exponential size.

References

1 Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 599–610. ACM, 2011.

2 Rajeev Alur, Antoine Durand-Gasselin, and Ashutosh Trivedi. From monadic second-order definable string transformations to transducers. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS)*, pages 458–467. IEEE Computer Society, 2013.

3 Mikolaj Bojańczyk. Transducers with origin information. In *Proceedings of the 41st International Colloquium, ICALP 2014*, LNCS, pages 26–37. Springer, 2014.

4 Olivier Carton and Luc Dartois. Aperiodic two-way transducers and FO-transductions. In *24th EACSL Annual Conference on Computer Science Logic (CSL)*, LIPIcs, pages 160–174. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.

5 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2(2):216–254, April 2001.

6 Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From two-way to one-way finite state transducers. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 468–477. IEEE Computer Society, 2013.

7 Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. First-order definable string transformations. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 29 of *LIPIcs*, pages 147–159, 2014.

8 M Lothaire. *Combinatorics on words*. Cambridge University Press, 1997.

9 Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

10 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

What's Decidable about Availability Languages?

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[1], Roland Meyer[2], and Mehdi Seyed Salehi[3]

1 **Uppsala University, Uppsala, Sweden**
{parosh,mohamed_faouzi.atig}@it.uu.se
2 **University of Kaiserslautern, Kaiserslautern, Germany**
meyer@cs.uni-kl.de
3 **Sharif University of Technology, Tehran, Iran**
seyedsalehi@ce.sharif.edu

Abstract

We study here the algorithmic analysis of systems modeled in terms of availability languages. Our first main result is a positive answer to the emptiness problem: it is decidable whether a given availability language contains a word. The key idea is an inductive construction that replaces availability languages with Parikh-equivalent regular languages. As a second contribution, we solve the intersection problem modulo bounded languages: given availability languages and a bounded language, it is decidable whether the intersection of the former contains a word from the bounded language. We show that the problem is NP-complete. The idea is to reduce to satisfiability of existential Presburger arithmetic. Since the (general) intersection problem for availability languages is known to be undecidable, our results characterize the decidability border for this model. Our last contribution is a study of the containment problem between regular and availability languages. We show that safety verification, i.e., checking containment of an availability language in a regular language, is decidable. The containment problem of regular languages in availability languages is proven undecidable.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.4 Mathematical Logic and Formal Languages

Keywords and phrases Availability, formal languages, emptiness, decidability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.192

1 Introduction

Availability is an important concept in the dependability analysis of unreliable reactive systems. In such systems, components may fail for some time and recover later. In other words, the system may be available for a certain amount of time during the observation period. The property of interest for us in such systems is (interval) availability which specifies the proportion of the time in which the system is available for use.

Availability for continuous systems has been extensively studied in the literature [4, 16]. Studying availability over a discrete domain, however, is quite new, but it can be useful. With appropriate approximations, it should be possible to model the availability characteristics of a system. With a discrete domain at hand, one may hope for automated analyses. This paper can be understood as providing evidence for the latter claim.

Regular availability expressions have been introduced in [11] as a model for discrete availability aspects. Availability expressions extend regular expressions by an additional operator. This so-called occurrence constraint associates a positive or negative availability

with the symbols of the alphabet and poses requirements on the accumulated availability of words. To give an example, the availability expression

$$((up + down)^* \checkmark)_{\#up - \#down > 0}$$

requires a system uptime of at least 50%. To achieve this, the semantics evaluates the occurrence constraint $\#up - \#down > 0$ every time the execution meets a symbol $\checkmark$.

Algorithmic analysis looks at system models from a language-theoretic point of view. Here, availability expressions are interesting because their language class is incomparable with basic classes like context-free languages. Therefore, language-specific properties like emptiness or intersection are non-trivial for them.

The earlier work [11] proved the undecidability of checking whether the intersection of two availability languages is empty or not. The *emptiness problem* itself, however, remained open. Emptiness may be considered the key problem in algorithmic verification: correctness requirements are typically stated as $\mathcal{L}_1 \subseteq \mathcal{L}_2$ and then rephrased as $\mathcal{L} = \emptyset$ with $\mathcal{L} = \mathcal{L}_1 \cap \overline{\mathcal{L}_2}$. The first contribution of this article is a positive answer to the emptiness problem of regular availability languages. We provide an algorithm that takes an availability expression *rae* and decides whether the associated language $\mathcal{L}(rae)$ is empty.

Technically, we show that availability languages have regular approximations that are exact with respect to emptiness. More precisely, given *rae* we construct a regular expression *reg* with the same Parikh image [14]: $\Pi(\mathcal{L}(rae)) = \Pi(\mathcal{L}(reg))$. The idea is to proceed by induction on the structure of availability expressions. In the base case, we directly construct a one-counter automaton M that captures the language of an expression $(reg)_{cstr}$. (Here, *cstr* is an occurrence constraint as in the above example.) With Parikh's result, we then obtain a regular language that is Parikh-equivalent with $\mathcal{L}(M)$. In the induction step, we apply this result to iteratively replace availability expressions by regular expressions. This leads to an algorithm for solving the emptiness problem with a non-elementary complexity. This is due to the exponential blow-up encountered, at each induction step, when computing the Parikh image of a one-counter automaton.

Our second contribution is a refined study of the (undecidable) intersection problem. Rather than inspecting the full intersection $\mathcal{L}(rae_1) \cap \mathcal{L}(rae_2)$, we restrict the search to words from a given bounded expression $bl = w_1^* \dots w_m^*$. This under-approximate verification technique is also known as pattern-based verification [6], and generalizes the popular idea of bounded countext switching [15]. Our finding is that the problem is only NP-complete.

Technically, we give a reduction to satisfiability of existential Presburger arithmetic [17]. We first rewrite $\mathcal{L}(rae_1) \cap \mathcal{L}(rae_2) \cap \mathcal{L}(bl)$ as $(\mathcal{L}(rae_1) \cap \mathcal{L}(bl)) \cap (\mathcal{L}(rae_2) \cap \mathcal{L}(bl))$, a trick due to Ginsburg and Spanier [9]. Then we show how to capture the latter intersection by a formula $\exists \tilde{x}.\varphi_1(\tilde{x}) \wedge \varphi_2(\tilde{x})$ with $\tilde{x} = x_1, \dots, x_m$. Intuitively, the task of φ_k with $k = 1, 2$ is to count the occurrences of w_1 to w_m in the intersection $\mathcal{L}(rae_k) \cap \mathcal{L}(bl)$.

The actual challenge in the proof is to construct φ_k. We again proceed by induction. To invoke the hypothesis, we guess the part $bl' = w_i^* \dots w_j^*$ of the bounded expression bl that will be traversed when rae_k passes through a top-level constraint $(rae')_{cstr'}$. To our surprise, parts of length one ($i = j$) were difficult to handle and needed an auxiliary construction. Such a part may be traversed multiple times. Hence, representing it by a Presburger formula would lead to non-linearity. We show how to compute a finite automaton that captures the language $\mathcal{L}(bl') \cap \mathcal{L}((rae')_{cstr'})$. Here, we need visibility arguments and study the boundedness behavior of the one-counter automata representing availability languages.

Our last contribution is a study of the containment problem between regular languages and availability languages. First, we show the decidability of $\mathcal{L}(rae) \subseteq \mathcal{L}(reg)$. Note that

this inclusion is a common formulation of safety verification problems. The proof is by a reduction to the emptiness problem of the language $\mathcal{L}(rae) \cap \overline{\mathcal{L}(reg)}$. As key argument, we establish closure of the class of availability languages under regular intersection. Second, we show the undecidability of checking whether a regular language is included in an availability language. The proof is by a reduction from the halting problem for two-counter automata (which is known to be undecidable [13]).

Related Work. We already discussed the relation of our results to availability analysis. We now concentrate on the related work in formal languages and verification. The work [11] introduced regular availability expressions and proposed a corresponding automaton model that captures availability languages in an operational rather than a declarative way. Moreover, it gave a synthesis algorithm that determines a most liberal implementation of an availability requirement. The final result was the undecidability of intersection emptiness. We focus here on algorithmic problems of the latter form, and obtain positive results. We show the decidability of emptiness, NP-completeness of a restricted intersection problem, and decidability of safety verification. We believe that these positive results, in particular the low complexity of the intersection problem modulo bounded languages, should motivate further studies of availability languages. It should also be noted that we generalize the model [11] towards stronger occurrence constraints.

Availability expressions introduce occurrence constraints to influence the use of symbols. With this numeric aspect, Parikh images [14] proved to be a valuable tool in the manipulation of availability languages. There are other language-theoretic models that employ Parikh images [12, 3, 2, 1, 18]. In all these models (variants of so-called Parikh automata and Presburger regular expressions), the final acceptance of a word depends on the number of occurrences of letters. What is different in our model is that we admit intermediary occurrence checks. These checks can be used as guards to influence the future system behavior, as opposed to post-mortem acceptance checks. There is no bound on the number of such intermediary measurements so that there is no immediate reduction to the aforementioned models.

Concerning bounded languages, Ganty et al. [8] showed how to construct from a context-free language a context-free bounded language with the same Parikh image. Also in the context-free setting, Esparza and Ganty proposed the intersection problem modulo bounded languages [6], a work that inspired our second contribution. Hague and Lin generalized this result to pushdown automata with reversal-bounded counters [10]. Our underlying model is regular rather than context free but admits an unbounded number of checks on the counters.

Weighted languages form another line of related work [5]. The idea is to let an automaton manipulate weights from a semi-ring. Actually, weighted automata admit very general semi-rings while we focus on the occurrence of letters. In contrast, our occurrence constraints can influence the system behavior while weighted automata only provide an analysis.

2 Availability Languages

Regular availability expressions extend regular expressions by occurrence constraints on the letters. The model was introduced in [11] and is presented here in a generalized form. The idea of occurrence constraints is to require a specified ratio on the occurrence of letters. For example, the word $w = aab$ has more letters a than b, which means the occurrence constraint $\#a - \#b > 0$ holds. To be more precise, the occurrence constraint is checked at the places marked by a distinguished symbol ✓. It does not need to hold throughout the word. With

this, the availability expression $(a^*b^*\checkmark)_{\#a-\#b>0}$ denotes the language $\{a^n b^m \mid n > m\}$. Throughout the paper, we assume an underlying finite alphabet Λ.

An *occurrence constraint cstr* takes the form

$$t + \sum_{a\in\Lambda} k_a \cdot \#a > 0 \qquad \text{with } t, k_a \in \mathbb{Z}. \tag{1}$$

The set of *regular availability expressions* is defined as follows:

$$rae \;::=\; a \mid \varepsilon \mid \emptyset \mid \checkmark \mid rae + rae \mid rae.rae \mid rae^* \mid (rae)_{cstr} \;.$$

Here, $a \in \Lambda$, $\checkmark \notin \Lambda$ is the distinguished symbol, and *cstr* is defined as above. We use *reg* to indicate that an availability expression actually is a *regular expression*, which means it does not contain $\checkmark$ nor *cstr*. The *depth* of an availability expression is the nesting depth of occurrence constraints $(rae)_{cstr}$. An occurrence constraint $(rae)_{cstr}$ is *top-level* in rae' if it is not covered by another $(-)_{cstr'}$ in the syntax tree of rae'. The syntactic *size* of an availability expression *rae* is denoted by $|rae|$. The definition is as expected, every piece of syntax contributes to it.

The semantics of availability expressions is in terms of finite words, $\mathcal{L}(rae) \subseteq (\Lambda \cup \{\checkmark\})^*$, and defined inductively as follows:

$$\begin{array}{lll}
\mathcal{L}(a) := \{a\} & \mathcal{L}(\checkmark) := \{\checkmark\} & \mathcal{L}(\varepsilon) := \{\varepsilon\} \\
\mathcal{L}(\emptyset) := \emptyset & \mathcal{L}(rae_1 + rae_2) := \mathcal{L}(rae_1) \cup \mathcal{L}(rae_2) & \mathcal{L}(rae^*) := \mathcal{L}(rae)^* \\
\mathcal{L}(rae_1.rae_2) := \mathcal{L}(rae_1).\mathcal{L}(rae_2) & \mathcal{L}(rae_{cstr}) := \mathcal{L}(rae)_{cstr}. &
\end{array}$$

To define the semantics of occurrence constraints, $\mathcal{L}(rae)_{cstr}$, we need Parikh images and projections. The *Parikh image* of a word w over $\Lambda \cup \{\checkmark\}$ is a function $\Pi(w) : \Lambda \to \mathbb{N}$ that returns for every $a \in \Lambda$ the number of occurrences of a in w, in symbols $\Pi(w)(a) := |w|_a$. Let $\Gamma \subseteq \Lambda \cup \{\checkmark\}$ and let w be a word over $\Lambda \cup \{\checkmark\}$. The *projection* of w to Γ, denoted by $\pi_\Gamma(w)$, is the result of removing all symbols outside Γ from w. Using these concepts, operator $\mathcal{L}_{cstr}$ checks that each prefix ending in $\checkmark$ satisfies the occurrence constraint, and projects the remaining words to Λ:

$$\begin{aligned}
\mathcal{L}_{cstr} &:= \{\pi_\Lambda(w) \mid w \in \mathcal{L}^{\checkmark}_{cstr}\} \\
\mathcal{L}^{\checkmark}_{cstr} &:= \{w \in \mathcal{L} \mid t + \sum_{a\in\Lambda} k_a \cdot \Pi(w_1)(a) > 0 \text{ for all } w_1.\checkmark.w_2 = w\} \;.
\end{aligned}$$

Availability languages are incomparable with context-free languages [11]. For example, the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ can be generated by the regular availability expression

$$((((a^*b^*c^*\checkmark)_{1+\#a-\#b>0}\checkmark)_{1+\#b-\#a>0}\checkmark)_{1+\#c-\#a>0}\checkmark)_{1+\#a-\#c>0}$$

but it is not a context-free language. The language of words $w.w^{reverse}$ in turn cannot be represented by a regular availability expression.

3 Emptiness

The *emptiness problem* for availability languages consists in checking, for a given regular availability expression *rae*, whether $\mathcal{L}(rae) = \emptyset$. Our first main result is the decidability of the emptiness problem. The solution is inductive. We first discuss the emptiness problem for

$(reg)_{cstr}$. Via one-counter automaton, we show how to obtain a Parikh-equivalent regular expression. Then we use this result to turn a general availability expression *rae* into a Parikh-equivalent regular expression *reg*: $\Pi(\mathcal{L}(rae)) = \Pi(\mathcal{L}(reg))$. With this correspondence, the two languages have the same emptiness status: $\mathcal{L}(rae) = \emptyset$ iff $\mathcal{L}(reg) = \emptyset$.

3.1 One-Counter Automata

We use a variant of one-counter automata (1CM) with counters over the integers $\mathbb{Z}$ rather than the natural numbers $\mathbb{N}$. A 1CM is a non-deterministic finite state automaton equipped with a counter that can be incremented, decremented, and compared to zero. Formally, the automaton is a 5-tuple $M = (Q, \Lambda, \Delta, s, F)$ where Q is a finite set of states with initial state $s \in Q$ and final states $F \subseteq Q$. The transitions in $\Delta \subseteq Q \times (\Lambda \cup \{\varepsilon\}) \times Op \times Q$ are labelled by a letter from Λ and equipped with an operation from $Op := \{> 0, \leq 0\} \cup \{\mathsf{add}(m) \mid m \in \mathbb{Z}\}$. For the semantics, we define labelled transitions between configurations from $Q \times \mathbb{Z}$. The automaton accepts a word $w \in \Lambda^*$ if there is a sequence of configurations that is labelled by w and ends in a final state. The language $\mathcal{L}(M)$ is the set of all words accepted by M.

A 1CM M over the integers can be compiled down to a language-equivalent 1CM $M_{\mathbb{N}}$ over the natural numbers. To adjust the semantics, over the naturals an addition $\mathsf{add}(-m)$, $m \in \mathbb{N}$, is enabled only if the resulting counter value stays at least zero. The idea of the translaton is to duplicate each control state. So state q in M yields q_+ and q_- in $M_{\mathbb{N}}$. In q_+, the counter value represents a positive value in the original automaton. In q_-, the value represents the corresponding negative value in the original automaton. Clearly, a transition > 0 from q will be copied to q_+ and will be removed from q_-. A transition ≤ 0 from q will be copied to q_+. In q_-, we have the transition without a condition (represented by $\mathsf{add}(0)$). Consider a decrement $(q, a, \mathsf{add}(-m), q')$, $m \in \mathbb{N}$. Besides the corresponding transitions at q_+ and q_-, every pair $x, y \in \mathbb{N}$ so that $x + y = m$ yields a transition sequence $(q_+, a, \mathsf{add}(-x), p)(p, \varepsilon, \leq 0, p')(p', \varepsilon, \mathsf{add}(y), q'_-)$ where p, p' are two intermediary states used only in the simulation of the decrement transition. The construction is similar for increments.

The 1CM $M_{\mathbb{N}}$ in turn can be understood as pushdown automata with two stack symbols. One of them is used to mark the bottom of the stack, the other represents the counter value. As a consequence of the two constructions, the languages of 1CM over $\mathbb{Z}$ are context free. With Parikh's theorem, we can construct a regular language with the same Parikh image. A simple construction which takes a context-free language and returns a Parikh-equivalent finite automaton has been proposed in [7]. We summarize the argumentation.

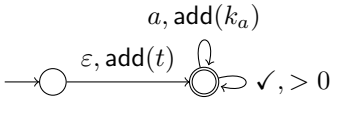
▶ **Lemma 3.1.** *Given a 1CM M, one can construct reg with $\Pi(\mathcal{L}(M)) = \Pi(\mathcal{L}(reg))$.*

1CM (over $\mathbb{Z}$ and over $\mathbb{N}$) are effectively closed under regular intersection and projection.

▶ **Lemma 3.2.** *Given 1CM M and reg, we can construct M' with $\mathcal{L}(M') = \mathcal{L}(M) \cap \mathcal{L}(reg)$. Given $\Gamma \subseteq \Lambda$, we can construct M' with $\mathcal{L}(M') = \pi_\Gamma(\mathcal{L}(M))$.*

3.2 From Availability to Regular Expressions

To check emptiness of $(reg)_{cstr}$, we first construct a 1CM M that accepts words over $\Lambda \cup \{\checkmark\}$ only based on the constraint *cstr*. With the notation from the previous section, it accepts all $w \in (\Lambda \cup \{\checkmark\})^*$ such that $\{w\}^{\checkmark}_{cstr} = \{w\}$. Assume *cstr* takes the Form (1). Automaton M, depicted to the right, has two states s and f, the former being initial and the latter being final, respectively. There is an ε-labelled transition from s



to f with operation $\mathsf{add}(t)$. The transition initializes the counter with constant t from the constraint. For every letter $a \in \Lambda$, we have a loop at state f that adds coefficient $k_a \in \mathbb{Z}$. Moreover, for ✓ we have a loop at f with check > 0.

▶ **Lemma 3.3.** *Let $w \in (\Lambda \cup \{\checkmark\})^*$. Then $w \in \mathcal{L}(M)$ if and only if $\{w\}^{\checkmark}_{cstr} = \{w\}$.*

To take the regular expression *reg* into account, we use closure under regular intersection. So given *reg* and M above, we compute M' with $\mathcal{L}(M') = \mathcal{L}(M) \cap \mathcal{L}(reg)$. To remove symbol ✓, we project $\mathcal{L}(M')$ to Λ. This yields the language of $(reg)_{cstr}$.

▶ **Lemma 3.4.** $\pi_\Lambda(\mathcal{L}(M')) = \mathcal{L}((reg)_{cstr})$.

By closure under projection, $\mathcal{L}((reg)_{cstr})$ is again a one-counter language. With Lemma 3.1, the language can be represented by a regular language, up to Parikh-equivalence.

▶ **Proposition 3.5.** *One can construct reg' with $\Pi(\mathcal{L}((reg)_{cstr})) = \Pi(\mathcal{L}(reg'))$.*

For the general case, consider *rae* and focus on an occurrence constraint of maximal depth. By maximality, it has the shape $(reg)_{cstr}$. We apply Proposition 3.5 to construct a regular language reg' with the same Parikh image, $\Pi(\mathcal{L}(reg')) = \Pi(\mathcal{L}((reg)_{cstr}))$. We now replace $(reg)_{cstr}$ by reg' within *rae*, resulting in rae'. The languages of *rae* and rae' still coincide, up to Parikh-equivalence:

$$\Pi(\mathcal{L}(rae')) = \Pi(\mathcal{L}(rae)).$$

The reason is that the Parikh image of $\mathcal{L}((reg)_{cstr})$ keeps the number of symbols (but may not retain their order), and this is what is needed to evaluate further occurrence constraints in *rae*. We repeat the procedure inductively on rae'. Eventually, we have eliminated all occurrence constraints and hence arrived at a regular expression.

▶ **Theorem 3.6.** *One can construct reg with $\Pi(\mathcal{L}(rae)) = \Pi(\mathcal{L}(reg))$. Hence, the emptiness problem for availability languages is decidable.*

Observe that the procedure has a non-elementary complexity. This is due to the exponential blow-up encountered, at each induction step, when computing the Parikh image of a 1CM.

4 Intersection Modulo Bounded Languages

The intersection problem $\mathcal{L}(rae_1) \cap \mathcal{L}(rae_2) \neq \emptyset$ is known to be undecidable, already for two availability expressions [11]. We now show that the problem remains decidable, actually only NP-complete, if we require the words in the intersection to belong to the language of a bounded expression of the form $bl = w_1^* \dots w_m^*$. To be precise, we study the following problem that we will refer to as IBL, *intersection modulo bounded languages*: Given rae_1 to rae_n and bl, is $\bigcap_{i=1}^n \mathcal{L}(rae_i) \cap \mathcal{L}(bl) \neq \emptyset$? Our second main result is as follows.

▶ **Theorem 4.1.** IBL *is* NP*-complete for availability expressions of fixed depth.*

We explain the proof approach and elaborate on the side condition. The approach is inspired by Ginsburg and Spanier's [9] and has also been used in [6]. We first rewrite the intersection:

$$\Big(\bigcap_{i=1}^{n} \mathcal{L}(rae_i)\Big) \cap \mathcal{L}(bl) \quad = \quad \bigcap_{i=1}^{n} \big(\mathcal{L}(rae_i) \cap \mathcal{L}(bl)\big)\,. \tag{2}$$

Let $bl = w_1^* \dots w_m^*$. Our technical contribution (Proposition 4.8) is then to compute in polynomial time an existential Presburger formula $\varphi_i(x_1, \dots, x_m)$ that captures the words in an intersection $\mathcal{L}(rae_i) \cap \mathcal{L}(bl)$ as follows: $w_1^{k_1} \dots w_m^{k_m} \in \mathcal{L}(rae_i) \cap \mathcal{L}(bl)$ if and only if $(k_1, \dots, k_m) \models \varphi_i$. Intuitively, the Presburger formula counts how often each word in the bounded expression occurs. With this, the intersection between the languages $\mathcal{L}(rae_i) \cap \mathcal{L}(bl)$ on the right-hand side of Equation (2) is represented by the intersection of the solution spaces of the corresponding formulas φ_i. Indeed, there is a word in the intersection if and only if there are coefficients k_1 to k_m on which all formulas agree. This is equivalent to satisfiability of $\exists x_1 \dots \exists x_m : \bigwedge_{i=1}^{n} \varphi_i(x_1, \dots, x_m)$. Since the formulas are computable in polynomial time (Proposition 4.8) and satisfiability of existential Presburger is in NP [17], we obtain an upper bound for IBL.

For the polynomial-time computability to hold, we have to assume the depth of the availability expressions given as input to be bounded from above by a constant. The need for a fixed depth comes with the proof approach. We construct the Presburger formulas by an induction on the depth of availability expressions. Each step in this induction is polynomial-time computable. The composition of the polynomials, however, only stays polynomial if we assume it to be fixed.

For the lower bound, NP-hardness already holds in the case of regular rather than availability languages, a single letter alphabet, and a fixed pattern, due to [6].

4.1 Bounded Languages and Presburger Arithmetic

In the literature, bounded languages are defined as (potentially non-regular) subsets of $w_1^* \dots w_m^*$. As there is no risk of confusion, we decided to adopt the terminology. For our proofs, we will have to deal with leading and trailing words. So we will also refer to $bl = w_0.w_1^* \dots w_m^*.w_{m+1}$ as a bounded expression. The *length* of bl is the number of starred words, $m \in \mathbb{N}$. A *part* of bl is a language $u.w_i^* \dots w_j^*.v$ with $1 \leq i \leq j \leq m$, u a suffix of w_{i-1} or w_i, and v a prefix of w_j or w_{j+1}. Alternatively, $u.v$ may form an infix of some w_i and the iterated part is missing.

Presburger arithmetic is the first-order logic of the natural numbers with addition but without multiplication. Given a formula $\varphi(x_1, \dots, x_n)$ with free variables x_1 to x_n, we use $\mathcal{S}(\varphi)$ for the solution space: the set of valuations $(k_1, \dots, k_n)$ that satisfy the formula. We are interested in the *existential fragment* of Presburger, denoted by ∃PA and defined by

$$t \ ::= \ 0 \mid 1 \mid x \mid t_1 + t_2 \qquad \varphi \ ::= \ t_1 = t_2 \mid t_1 > t_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi \ .$$

A result by Verma, Seidl, and Schwentick shows how to capture the Parikh images of context-free languages by existential Presburger.

▶ **Proposition 4.2** ([19]). *Given a context-free grammar G, one can compute in linear time an ∃PA formula φ satisfying $\mathcal{S}(\varphi) = \Pi(\mathcal{L}(G))$.*

4.2 NP Upper Bound

Our goal is to construct an ∃PA formula for $\mathcal{L}((rae)_{cstr}) \cap \mathcal{L}(bl)$. Roughly, the approach is to represent the intersection by a 1CM and then obtain the ∃PA formula with Proposition 4.2. More precisely, the construction is by induction on the depth of availability expressions, and the challenge is to handle the top-level occurrence constraints $(rae')_{cstr'}$ within $(rae)_{cstr}$. To invoke the hypothesis, the idea is to precompute the intersection of $(rae')_{cstr'}$ with parts bl' of the bounded language bl. This, however, requires care. We will have to treat parts of

length at most one and parts of length at least two substantially different. As a result, we will have to invoke different hypotheses. We explain the difficulties.

For parts of length at most one, the intersection $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$ may be entered several times. Indeed, bl' may be $u.w_i^*.v$ with u a suffix of w_i and v a prefix of w_i. If we only had a Presburger formula to represent the intersection, re-entering the intersection would lead to a non-linear constraint. Instead, we show (in a separate induction, leading to Proposition 4.3) how to compute a finite automaton representing the intersection. This automaton can then be plugged into the overall 1CM construction.

For parts $bl' = u.w_i^* \dots w_j^*.v$ of length at least two, we cannot re-enter an intersection $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$, simply because $j > i$. This allows us to represent the intersection by an ∃PA formula, which we obtain by a simple invocation of the induction hypothesis. We incorporate this formula into the ∃PA formula for the overall construction. Note that the intersection with a bounded language of length at least two is not necessarily a regular language. This means the automaton trick for length at most one does not work.

4.2.1 Automata for Length at most One

In the following, we show how to construct, in polynomial time, a finite state automaton recognizing the intersection of an availability expression and a part of a bounded expression of length at most one.

▶ **Proposition 4.3.** *Consider availability expressions of fixed depth. Given such an expression* $(rae)_{cstr}$ *and a bounded language* $bl = w_1.w^*.w_2$*, one can compute in polynomial time an NFA* A *with* $\mathcal{L}(A) = \mathcal{L}((rae)_{cstr}) \cap \mathcal{L}(bl)$*.*

The rest of this section is devoted to the proof of Proposition 4.3 which is done by induction on the depth of the availability expression.

Base Case. We first prove the base case when the availability expression is actually a regular expression (see Lemma 4.4).

▶ **Lemma 4.4.** *Given* $(reg)_{cstr}$ *and* $bl = w_1.w^*.w_2$*, one can compute in polynomial time an NFA* A *with* $\mathcal{L}(A) = \mathcal{L}((reg)_{cstr}) \cap \mathcal{L}(bl)$*.*

To prove Lemma 4.4, we first construct a 1CM for the intersection of the languages and in a second step compile this automaton down to a finite automaton. In Section 3, we have shown how to turn $(reg)_{cstr}$ into a 1CM with the same language. Since 1CM are closed under regular intersection, we can also determine $M = (Q, \Lambda, \Delta, s, F)$ with $\mathcal{L}(M) = \mathcal{L}((reg)_{cstr}) \cap \mathcal{L}(bl)$. Note that the construction of M works in polynomial time. To turn M into a finite automaton, the key observation is that M satisfies the following property referred to as **(Bound)**. There is a constant $b \in \mathbb{N}$ so that
(Bound-U) once we exceed b in a configuration, the counter will never drop below zero,
(Bound-L) once we fall below $-b$, the counter will not increase above zero again.
For the formalization, we focus on **(Bound-U)**, **(Bound-L)** is similar:

$$\forall (q,c) \in Q \times \mathbb{Z} \text{ with } (s,0) \to^* (q,c) \text{ and } c > b : \\ \forall (q',c') \in Q \times \mathbb{Z} \text{ with } (q,c) \to^* (q',c') : \quad c' > 0.$$

The following, lemma shows that indeed the 1CM M satisfies the property **(Bound)** .

▶ **Lemma 4.5.** M *satisfies* **(Bound)** *with* $b \in \mathbb{N}$ *of size polynomial in* $|rae|+|bl|$*.*

We defer the proof of Lemma 4.5 for a moment and show that property **(Bound)** implies Lemma 4.4: the correspondingly bounded 1CM accept regular languages. The reason is that we only have to track the counter value precisely as long as it stays in the interval $[-b, b]$. Once this range is left, **(Bound-L)** and **(Bound-U)** indicate how to evaluate guards.

▶ **Lemma 4.6.** *Assume 1CM M satisfies* **(Bound)** *with $b \in \mathbb{N}$. There is an NFA A of size polynomial in $|M|+b$ with $\mathcal{L}(A) = \mathcal{L}(M)$.*

To prove Lemma 4.5, recall that $\mathcal{L}(M) = \mathcal{L}(bl) \cap \mathcal{L}((reg)_{cstr})$ with $bl = w_1.w^*.w_2$ and $cstr$ of the Form (1). The main observation is that M is a *visibly* 1CM in the following sense. A letter $a \in \Lambda$ always has the effect of adding the coefficient $k_a \in \mathbb{Z}$ to the counter, independent of the transition. This means no matter which transition sequence the automaton takes to process the word w in $bl = w_1.w^*.w_2$, the effect on the counter is always constant. We refer to it as $\mathit{effect}(w) \in \mathbb{Z}$. Note that the effect is homomorphic, $\mathit{effect}(w.v) = \mathit{effect}(w) + \mathit{effect}(v)$. We then do a case distinction according to whether w has a positive or a negative effect. Assume $\mathit{effect}(w) \geq 0$. We define the constant $b \in \mathbb{N}$ to be $|t|+max\{|\mathit{effect}(u)| \mid u$ an infix of $w_a.w_b$ where $w_a \in \{w_1, w\}$ and $w_b \in \{w, w_2\}\}$.

For **(Bound-L)**, we show that the counter never drops below $-b$ and hence the property trivially holds. For **(Bound-U)**, we consider a configuration with counter value $c > b$ and argue that the value stays above zero in every continuation of the transition sequence.

Induction step. Next, we show the induction step for the proof of Proposition 4.3. The induction step is established using the following lemma:

▶ **Lemma 4.7.** *Assume Proposition 4.3 holds for availability expressions of depth at most $n \in \mathbb{N}$. Consider $(rae)_{cstr}$ of depth $n+1$ and $bl = w_1.w^*.w_2$. One can compute in polynomial time an NFA A with $\mathcal{L}(A) = \mathcal{L}((rae)_{cstr}) \cap \mathcal{L}(bl)$.*

Proof. The idea is to consider every part of the bounded expression $w_1.w^*.w_2$ that may be traversed when $(rae)_{cstr}$ passes through a top-level occurrence constraint. For example, $\mathcal{L}((rae)_{cstr}) \cap \mathcal{L}(bl)$ may traverse $bl' = u.w^*.v$ while being in $(rae')_{cstr'}$, with u a suffix of w and v a prefix of w. Each part bl' is again a bounded expression of the form assumed by the lemma. This means for each combination of top-level constraint $(rae')_{cstr'}$ and part bl', we can apply the hypothesis and compute a finite automaton $A_{(rae')_{cstr'}, bl'}$ representing the intersection. We now modify the given availability expression $(rae)_{cstr}$ to $(\tilde{rae})_{\tilde{cstr}}$ by replacing every top-level constraint $(rae')_{cstr'}$ with $\bigcup_{bl' \text{ part of } bl} A_{(rae')_{cstr'}, bl'}$. This replacement is sound and complete in the sense that

$$\mathcal{L}((rae)_{cstr}) \cap \mathcal{L}(bl) \;=\; \mathcal{L}((\tilde{rae})_{\tilde{cstr}}) \cap \mathcal{L}(bl) \;. \tag{3}$$

Soundness holds by $\mathcal{L}(A_{(rae')_{cstr'}, bl'}) = \mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl') \subseteq \mathcal{L}((rae')_{cstr'})$. Completeness is because we consider every part of bl. The finite automaton of interest is constructed from the right-hand side of Equation (3) by going through 1CM, as in Lemma 4.4. Note that $(\tilde{rae})_{\tilde{cstr}}$ indeed has the form $(reg)_{\tilde{cstr}}$ so that the argument from the base case applies.

To see that the construction is polynomial time, note that the number of parts of bl is quadratic. We avoid computing regular expressions for the automata $A_{(rae')_{cstr'}, bl'}$ but directly incorporate them into the 1CM construction. ◀

4.2.2 Presburger for the General Case

With the previous automaton construction at hand, we are now prepared to address our actual goal: computing an ∃PA formula that characterizes the intersection of a regular availability language with a bounded language. Our main result is the following proposition:

▶ **Proposition 4.8.** *Consider availability expressions of fixed depth. Given such an expression* $(rae)_{cstr}$ *and a bounded language* $bl = w_0.w_1^* \dots w_m^*.w_{m+1}$*, one can compute in polynomial time an* $\exists PA$ *formula* $\varphi(x_1, \dots, x_m)$ *so that for all* $k_1, \dots, k_m \in \mathbb{N}$*:*

$$(k_1, \dots, k_m) \models \varphi \qquad \text{if and only if} \qquad w_0.w_1^{k_1} \dots w_m^{k_m}.w_{m+1} \in \mathcal{L}((rae)_{cstr}) \cap \mathcal{L}(bl).$$

The rest of this section is dedicated to the proof of Proposition 4.8. The proof is done by induction on the depth of the availability expression.

Base Case. We first prove the base case when the availability expression is actually a regular expression (see Lemma 4.9).

▶ **Lemma 4.9.** *Given* $(reg)_{cstr}$ *and* $bl = w_0.w_1^* \dots w_m^*.w_{m+1}$*, one can compute in polynomial time an* $\exists PA$ *formula* $\varphi(x_1, \dots, x_m)$ *as required.*

We introduce fresh letters to the bounded language: $bl' := w_0.(w_1.a_1)^* \dots (w_m.a_m)^*.w_{m+1}$. Now an occurrence of a_i signals a full occurrence of w_i. We compute the product with the 1CM for $(reg)_{cstr}$ and apply Proposition 4.2. It yields an $\exists$PA formula which, after existential quantification of the variables for the original letters, is as required by Lemma 4.9.

Induction Step. Next, we show the induction step for the proof of Proposition 4.8. The induction step is established using the following lemma.

▶ **Lemma 4.10.** *Assume Proposition 4.8 holds for availability expressions of depth at most* $n \in \mathbb{N}$*. Consider* $(rae)_{cstr}$ *of depth* $n+1$ *and* $bl = w_0.w_1^* \dots w_m^*.w_{m+1}$*. One can compute in polynomial time an* $\exists PA$ *formula* $\varphi(x_1, \dots, x_m)$ *as required.*

This is the proof where we need the two hypotheses: that we can compute a finite automaton representing an intersection with a bounded language of length at most one, and that we can construct an $\exists$PA formula characterizing an intersection with a bounded language of length at least two. We shall assume $m \geq 2$, for otherwise we can apply Lemma 4.9 to the automaton from Proposition 4.3.

Proof. We first define a modification of the given availability expression. It will involve adding new letters that indicate the occurrence of an intersection with a bounded language of length at least two. In a following step, we turn the bounded expression into a finite automaton that takes the fresh letters into account. Then we determine the $\exists$PA formula of interest. The proof concludes with an estimation of the complexity.

Modifying the availability expression Consider every top-level constraint $(rae')_{cstr'}$. For every part bl' of the bounded language that has length at most one, we apply Proposition 4.3. It yields a finite automaton $A_{(rae')_{cstr'}, bl'}$ with language $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$. Moreover, for every part bl' of length at least two, we introduce a fresh letter $\sharp_{(rae')_{cstr'}, bl'}$. We now replace $(rae')_{cstr'}$ by the regular language

$$\bigcup_{\substack{bl' \text{ part of } bl \\ \text{of length} \leq 1}} A_{(rae')_{cstr'}, bl'} \quad \cup \quad \bigcup_{\substack{bl' \text{ part of } bl \\ \text{of length} > 1}} \sharp_{(rae')_{cstr'}, bl'} \ .$$

The modified availability expression is the result of all these replacements.

Turning the bounded expression into a finite automaton An occurrence of $\sharp_{(rae')_{cstr'}, bl'}$ will represent an occurrence of $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$. The task of the automaton associated with bl is to enforce that such an intersection is not traversed twice. The construction is

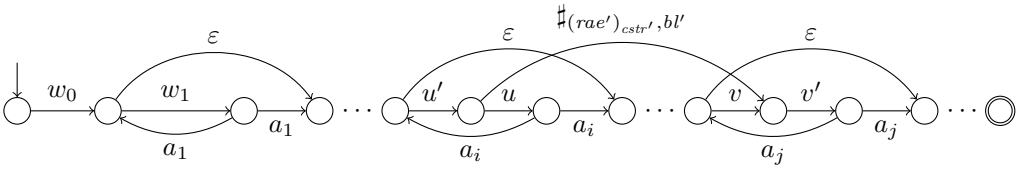


Figure 1 Illustration of A_{bl}.

illustrated in Figure 1. First, we introduce fresh letters counting w_1 to w_m. This gives $w_0.(w_1.a_1)^* \dots (w_m.a_m)^*.w_{m+1}$. When represented as a finite automaton, we have a state q_w for every prefix w of $w_0.w_1.a_1 \dots w_{m+1}$. Consider the part $bl' = u.w_i^* \dots w_j^*.v$ with u a suffix of $w_i = u'.u$ and v a prefix of $w_j = v.v'$. For every top-level availability constraint $(rae')_{cstr'}$ we add a transition labelled by $\sharp_{(rae')_{cstr'},bl'}$ from $q_{w_0 \dots u'}$ to $q_{w_0 \dots v}$. Note that such a transition can be taken only once. The result is A_{bl}.

Computing the ∃PA formula The modification of the given availability expression $(rae)_{cstr}$ is of the form $(reg)_{cstr}$. We turn it into a 1CM and add loops to all states to guess the occurrences of the fresh letters a_1 to a_m. Let the result be M. Since A_{bl} is a finite automaton and 1CM are closed under regular intersection, we can compute the product $M \times A_{bl}$. We turn this product into a context-free grammar and apply Proposition 4.2. This gives us an ∃PA formula of the form $\psi(\tilde{p}, \tilde{y}, \tilde{z})$. The vectors of variables are as follows:

$\tilde{p}$ has one variable p_a for every letter $a \in \Lambda$,
$\tilde{y}$ has one variable y_i for every word $w_i \in \{w_1, \dots, w_m\}$,
$\tilde{z}$ has one variable $z_\sharp$ for each $\sharp = \sharp_{(rae')_{cstr'},bl'}$, $(rae')_{cstr'}$ top-level, and bl' of length ≥ 2.

Let $\sharp$ refer to a pair of top-level occurrence constraint $(rae')_{cstr'}$ and part of length at least two $bl' = u.w_i^* \dots w_j^*.v$. By the hypothesis, we can compute an ∃PA formula $\psi_\sharp(x_i, \dots, x_j)$ for $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$. We modify the formula to $\varphi_\sharp(z_\sharp, x_1^\sharp, \dots, x_m^\sharp)$:

$$\exists x_i \dots \exists x_j \ : \ \left(z_\sharp = 0 \ \wedge \bigwedge_{k=1}^{m} x_k^\sharp = 0 \right) \ \vee$$
$$\left(z_\sharp = 1 \ \wedge \bigwedge_{k<i \vee k>j} x_k^\sharp = 0 \ \wedge \ x_i^\sharp = x_i + 1 \ \wedge \bigwedge_{i<k\leq j} x_k^\sharp = x_k \ \wedge \ \psi_\sharp(x_i, \dots, x_j) \right).$$

If $z_\sharp$ is zero, the transition in A_{bl} labelled by $\sharp$ is not taken. This means the intersection $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$ does not contribute to the occurrences of w_1 to w_m. Therefore, we require the fresh variables $x_k^\sharp$ to be zero. If $z_\sharp$ is not zero, it has to be one because the $\sharp$-labelled transition can be taken at most once. Since $bl' = u.w_i^* \dots w_j^*.v$, the intersection $\mathcal{L}((rae')_{cstr'}) \cap \mathcal{L}(bl')$ still does not contribute to the occurrences of w_k with $k < i$ or $k > j$. For w_i we have $x_i + 1$ occurrences. The additional occurrence is for the suffix u. For w_k with $i < k \leq j$, we have precisely x_i occurrences of w_i. Note that we do not have to count the half occurrence of v. Automaton A_{bl} will later see an a_i signalling the occurrence of the composed $v.v' = w_j$.

With this, we can define the overall formula $\varphi(x_1, \dots, x_m)$:

$$\exists \tilde{p} \ \exists \tilde{y} \ \exists \tilde{z} \ \exists_\sharp \tilde{x}^\sharp \ : \ \bigwedge_{i=1}^{n} x_i = y_i + \sum_{\sharp} x_i^\sharp \quad \wedge \quad \bigwedge \varphi_\sharp(z_\sharp, x_1^\sharp, \dots, x_m^\sharp) \quad \wedge \quad \psi(\tilde{p}, \tilde{y}, \tilde{z}).$$

The formula sums up the occurrences of w_i in a new free variable x_i. These occurrences are given by y_i for the outer constraint and for the top-level occurrence constraints that are

intersected with a bounded language of length at most one. For the occurrences of w_i in an intersection with a bounded language of length at least two, we sum up the variables $x_i^\sharp$. Note that they are set to zero in case an intersection $\sharp$ is not taken. The remainder adds the formulas $\varphi_\sharp$ for the intersections $\sharp$ of top-level constraints with bounded languages of length at least two, and also adds the formula ψ for the overall intersection. Existential quantifiers hide all the auxiliary variables. We note that $\varphi_\sharp$ as well as ψ are existential Presburger formulas that may contain quantifiers. Since they are surrounded by conjunctions and disjunctions, the scope of these quantifiers can be extruded without harm.

Time complexity of the construction The automata representing the intersections with bounded languages of length at most one can be computed in polynomial time by Proposition 4.3. Similarly, the conversion of bl into A_{bl} can be computed in polynomial time. Indeed, the number of top-level occurrence constraints is bounded by the size of the input. Moreover, there is at most a quadratic number of pairs $w_i^* \dots w_j^*$ and again a quadratic number of prefixes and suffixes. Altogether, there is a polynomial number of symbols that we add. As a result, also the product of 1CM and A_{bl} can be computed in polynomial time, and similar for the Presburger formula $\psi(\tilde{p}, \tilde{y}, \tilde{z})$. It remains to add $\exists$PA formulas for a polynomial number of intersections $\sharp$. Each such formula can be determined in polynomial time by the hypothesis of the lemma. As a result, we have an overall polynomial time construction. ◀

One can optimize the construction by considering a more general notion of parts $U.w_i^* \dots .w_j^*.V$ where U is the union of all suffixes of w_i and w_{i-1} and V is the union of all prefixes of w_j and w_{j+1}. This does not change the overall complexity.

5 Containment

We study the problem of whether an availability language is contained in a regular language and vice versa. For the former problem, we show that availability languages are closed under regular intersection.

▶ **Theorem 5.1.** *Given rae and reg, we can construct rae′ with* $\mathcal{L}(rae') = \mathcal{L}(rae) \cap \mathcal{L}(reg)$. *With Theorem 3.6,* $\mathcal{L}(rae) \subseteq \mathcal{L}(reg)$ *is decidable.*

For the proof, we represent the regular language by a finite automaton. Then we compute, for each pair of entry and exit state, the intersection of the corresponding regular language with the top-level occurrence constraints. This gives an inductive construction.

For the reverse inclusion, we show the undecidability by a reduction from the halting problem for two-counter automata (2CM) [13]. 2CM are defined like the 1CM in Section 3 but use two counters. We can assume them to only add 1 or -1, and will use inc and dec, instead. So the overall alphabet is $\Lambda := \{\mathsf{inc}(i), \mathsf{dec}(i), \mathsf{zero}(i) \mid i = 1, 2\}$.

The idea of the reduction is to understand a 2CM as a finite automaton. The automaton only reflects the control-flow but does not take into account the semantics of counters. This means the language is regular, let it be $\mathcal{L}(reg)$. We define an availability language $\mathcal{L}(rae)$ that contains all words over Λ violating the semantics of two-counter automata. Together,

$$\mathcal{L}(reg) \subseteq \mathcal{L}(rae) \qquad \text{iff} \qquad \mathcal{L}(reg) \cap \overline{\mathcal{L}(rae)} = \emptyset.$$

Language $\mathcal{L}(reg) \cap \overline{\mathcal{L}(rae)}$ restricts the regular control-flow language to words respecting the semantics of counters. This language is empty if and only if the 2CM does not halt.

▶ **Theorem 5.2.** $\mathcal{L}(reg) \subseteq \mathcal{L}(rae)$ *is undecidable, even for rae of depth* 1.

Proof. It remains to define rae. The expression is a choice $rae := rae_1 + rae_2$ where rae_1 reflects the bad behavior on counter 1, and similar for rae_2. There are two choices for bad behavior: we decrement a counter below zero (see $rae_{1,1}$) or a test for zero fails (see $rae_{1,2}$):

$$rae_1 := (\Lambda^*.\mathsf{zero}(1) + \varepsilon).(rae_{1,1} + rae_{1,2}).\Lambda^*$$
$$rae_{1,1} := ((\ \mathsf{inc}(1) + \mathsf{dec}(1) + \mathsf{inc}(2) + \mathsf{dec}(2) + \mathsf{zero}(2)\ )^*.\checkmark)_{\#\mathsf{dec}(1)>\#\mathsf{inc}(1)}$$
$$rae_{1,2} := ((\ \mathsf{inc}(1) + \mathsf{dec}(1) + \mathsf{inc}(2) + \mathsf{dec}(2) + \mathsf{zero}(2)\ )^*.\checkmark.\mathsf{zero}(1))_{\#\mathsf{inc}(1)>\#\mathsf{dec}(1)}.$$

◀

6 Concluding Remarks and Future Work

Availability languages extend regular languages by occurrence constraints on the letters [11]. The extension increases expressiveness and leads to a class of languages incomparable with the context-free ones. In this paper, we contributed positive results to the algorithmic analysis of availability languages. Our first result is the decidability of the emptiness problem that was left open in [11]. Our solution is inductive and combines an explicit one-counter automata construction with Parikh's theorem. Our second result is NP-completeness of the intersection problem modulo bounded languages. The idea is to reduce to satisfiability of existential Presburger arithmetic. The reduction needs arguments about the boundedness behavior of the one-counter automata representing availability languages. Finally, we study regular containment. We obtain a positive result for safety verification $\mathcal{L}(rae) \subseteq \mathcal{L}(reg)$ and a negative result for the reverse inclusion $\mathcal{L}(reg) \subseteq \mathcal{L}(rae)$.

For future work, we see practical as well as theoretical avenues. On the practical side, we plan to study the use of availability languages in model checking. Although we have shown safety verification to be decidable, the question remains how to check the inclusion efficiently in practice. On the theoretical side, it should be beneficial to compare availability languages with other models. It would be attractive to have a uniform understanding of Parikh automata, Presburger languages, and availability languages. Extensions of monadic second-order logic designed to capture availability requirements would also be interesting. Finally, there is no omega-theory of availability.

References

1 M. Cadilhac, A. Finkel, and P. McKenzie. On the expressiveness of Parikh automata and related models. *arXiv:1101.1547 [cs]*, 2011.

2 M. Cadilhac, A. Finkel, and P. McKenzie. Affine Parikh automata. *RAI*, 46(4):511–545, 2012.

3 M. Cadilhac, A. Finkel, and P. McKenzie. Bounded Parikh automata. *International Journal of Foundations of Computer Science*, 23(8):1691–1709, 2012.

4 E. de Souza e Silva and H. R. Gail. Calculating availability and performability measures of repairable computer systems using randomization. *JACM*, 36(1):171–193, 1989.

5 M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs. Springer, 2009.

6 J. Esparza and P. Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL*, pages 499–510. ACM, 2011.

7 J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh's theorem: A simple and direct automaton construction. *IPL*, 111(12):614–619, 2011.

8 P. Ganty, R. Majumdar, and B. Monmege. Bounded underapproximations. *FMSD*, 40(2):206–231, 2012.

9 S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematic*, 16(2):285–296, 1966.

10 M. Hague and A. W. Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, volume 7358 of *LNCS*, pages 260–276. Springer, 2012.

11 J. Hoenicke, R. Meyer, and E.-R. Olderog. Kleene, Rabin, and Scott are available. In *CONCUR*, number 6269 in LNCS, pages 462–477. Springer, 2010.

12 F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In *ICALP*, volume 2719 of *LNCS*, pages 681–696. Springer, 2003.

13 M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

14 R. J. Parikh. On context-free languages. *JACM*, 13(4):570–581, 1966.

15 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

16 G. Rubino and B. Sericola. Interval availability distribution computation. In *Fault-Tolerant Computing*, pages 48–55, 1993.

17 B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the AMS*, 284(1):203–218, 1984.

18 H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *PODS*, pages 155–166. ACM, 2003.

19 K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In *CADE*, volume 3632 of *LNCS*, pages 337–352. Springer, 2005.

Towards Better Separation between Deterministic and Randomized Query Complexity*

Sagnik Mukhopadhyay and Swagato Sanyal

Tata Institute of Fundamental Research
Mumbai, India
sagnik@tifr.res.in, swagatos@tcs.tifr.res.in

Abstract

We show that there exists a Boolean function F which gives the following separations among deterministic query complexity ($D(F)$), randomized zero error query complexity ($R_0(F)$) and randomized one-sided error query complexity ($R_1(F)$): $R_1(F) = \widetilde{O}(\sqrt{D(F)})$ and $R_0(F) = \widetilde{O}(D(F))^{3/4}$. This refutes the conjecture made by Saks and Wigderson that for any Boolean function f, $R_0(f) = \Omega(D(f))^{0.753..}$. This also shows widest separation between $R_1(f)$ and $D(f)$ for any Boolean function. The function F was defined by Göös, Pitassi and Watson who studied it for showing a separation between deterministic decision tree complexity and unambiguous non-deterministic decision tree complexity. Independently of us, Ambainis *et al* proved that different variants of the function F certify optimal (quadratic) separation between $D(f)$ and $R_0(f)$, and polynomial separation between $R_0(f)$ and $R_1(f)$. Viewed as separation results, our results are subsumed by those of Ambainis *et al*. However, while the functions considered in the work of Ambainis *et al* are different variants of F, in this work we show that the original function F itself is sufficient to refute the Saks-Wigderson conjecture and obtain widest possible separation between the deterministic and one-sided error randomized query complexity.

1998 ACM Subject Classification F.1.1 [Computation by Abstract Devices]: Models of Computation – Relations between models, F.1.2 [Computation by Abstract Devices]: Modes of Computation – Probabilistic computation

Keywords and phrases Deterministic Decision Tree, Randomized Decision Tree, Query Complexity, Models of Computation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.206

1 Introduction

In computational complexity theory, one major goal is to prove limitations of existing computational models which helps us to understand the computational power that each model exhibits. Among the vast array of computational models that are studied in the literature, one of the simplest is query model (or decision tree model) where an algorithm computing a boolean function is given query access to the input. The algorithm queries different bits of the input, possibly in adaptive fashion, and computes the function on the input based on the query responses. The algorithm is charged not for the computation but for the number of bits it queries. It is easy to see that n is a trivial upper bound on the number of queries that any algorithm makes to evaluate the function where n is the input size. The objective is to minimize the number of queries made.

* S. Mukhopadhyay is supported by a TCS fellowship and S. Sanyal was supported by a DAE fellowship.

For a Boolean function f, the deterministic query complexity, $D(f)$, of f is defined to be the maximum (over inputs) number of queries the best deterministic query algorithm for f makes. For many well studied boolean functions, such as parity, threshold functions, the deterministic decision tree complexity is n - such functions are called *evasive* functions. As observed by Rivest and Vuillemin [7], most boolean functions are evasive. In this work we are mainly concerned about the power of the query model when we allow randomness. We want to ask the following question: how many queries can we save for evaluating f if we allow the query algorithm to toss coins. A randomized query algorithm can be thought of as a distribution over deterministic query algorithms. It can also be viewed as a query algorithm where each node has an additional power of tossing coins . After querying the variable associated with any internal node of the tree, the algorithm decides which input bit to query depending on the responses to the queries so far (i.e. the current node in the tree) and the value of the coin tosses while in that node. It is not hard to see that the two definitions are equivalent. We look at the following complexity measures that are well studied. The bounded-error randomized query complexity $R(f)$ of f is defined to be the number of queries made on the worst input by the best randomized query algorithm for f that is correct with probability $2/3$ on every input. The zero error randomized query complexity of f, $R_0(f)$, is the expected number of queries made on the worst input by the best randomized algorithm for f that gives correct answer on each input with probability 1. Finally the one-sided randomized query complexity of f, $R_1(f)$, is the number of queries made on the worst input by the best algorithm that is correct on every input with probability at least $2/3$, and in addition correct on every 0-input with probability 1. The choice of the constant $2/3$ in our definitions is arbitrary, and could be replaced by any constant in the definition of $R_1(f)$, and any constant greater than $1/2$ in the definition of $R(f)$, while changing the number of queries by a constant factor.

Relationships between these query complexity measures have been extensively studied in the literature. That randomization can save more than a constant factor of queries has been known for a long time. Snir [10] showed a $O(n^{\log_4 3})$ randomized linear query algorithm (a more powerful model than what we discussed) for complete binary NAND tree function for which the deterministic linear query complexity is $\Omega(n)$. Later on Saks and Wigderson [8] gave a $\Theta(n^{0.753\cdots})$ randomized query algorithm for the same function. They also showed that for uniform rooted ternary majority tree function, the randomized query complexity is $O(n^{0.893\cdots})$ and deterministic query complexity is $\Omega(n)$ - the authors credited R. Boppana for this example. All these example showed that randomized query complexity can be substantially lower than its deterministic counterpart.
In their paper, Saks and Wigderson made the following conjecture.

▶ **Conjecture 1** (Saks and Wigderson [8]). *For any boolean function f, $R_0(f) = \Omega(D(f)^{0.753\cdots})$.*

Saks and Wigderson conjectured that the complete binary NAND tree function exhibits the widest separation possible between these two measures of complexity. During this work, the best separation known between deterministic decision tree complexity and zero error, one-sided error and bounded error randomized query complexities was the one exhibited by the complete binary NAND tree function. Also, no separation among the different randomized query complexity measures was known.

For complete binary NAND tree function F, Santha [9] showed that $R(F) = (1-2\epsilon)R_0(F)$ where ϵ is the error probability. So, for this function, we have $R(F) = \Theta(D(F)^{0.753\cdots})$. It is easy to see that $D(f) \geq R(f), R_0(f), R_1(f)$. Blum and Impagliazzo [3], Tardos [11] and Hartmanis and Hemachandra [5] independently showed that $R_0(f) \geq \sqrt{D(f)}$. Nisan [6] showed that for any Boolean function f, $R(f) \geq \sqrt[3]{D(f)}/27$ and $R_1(f) \geq \sqrt{D(f)}$. The

biggest gap known so far between $D(f)$ and $R(f)$ for any f is much less than cubic and little progress has been made in last 20 years to improve the state of the art.

The main results of this work are Theorems 1 and 2. Theorem 1 refutes Conjecture 1 by Saks and Wigderson.

▶ **Theorem 1.** *There exists a Boolean function F for which $R_0(F) = \widetilde{O}(D(F)^{3/4})$.*

It is to be noted that this result does not match the lower bound of $R_0(f)$ in terms of $D(f)$. We also show quadratic separation between deterministic and one-sided randomized query complexity which is achieved by the same function.

▶ **Theorem 2.** *There exists a Boolean function F for which $R_1(F) = \widetilde{O}(\sqrt{D(F)})$.*

This separation matches the lower bound, upto logarithmic factors, on $R_1(f)$ in terms of $D(f)$ for any function f. These results give better separation between the corresponding complexity measures than what is known during this work.

The function F which yields these separation results was first introduced by Göös et al [4] for showing a gap between deterministic decision tree complexity and unambiguous non-deterministic decision tree complexity and resolving the famous *clique vs independent set* problem. We will define the function in Section 1.1.

Independently of us, Ambainis et al [2] proved various separation results between different query complexity measures. Among several other results, the authors prove the existence of a function f for which $R_0(f) = \widetilde{O}(\sqrt{D(f)})$. In view of the lower bound, this is the widest separation possible between these two measures. This also refutes the conjecture by Saks and Wigderson. Moreover, since $R_0(f) = \Omega(R_1(f))$, this also certifies the same separation as that of Theorem 2. However, the authors use a variant of the function F which was introduced by Göös et al [4] to show this separation. In our work, we showed that the original function F itself is sufficient to refute Saks-Wigderson conjecture and to show the widest possible separation between $D(f)$ and $R_1(f)$ for any boolean function f.

1.1 The Göös-Pitassi-Watson Function

We define the function F now. The domain of F is $\mathcal{D} = \{0,1\}^{n(1+\lceil \log n \rceil)}$. An input $M \in \mathcal{D}$ to F is viewed as a matrix of dimension $\sqrt{n} \times \sqrt{n}$. Each cell $M_{i,j}$ of M consists of two parts:

1. A *bit-entry* $b_{i,j} \in \{0,1\}$.
2. A *pointer-entry* $p_{i,j} \in \{0,1\}^{\lceil \log n \rceil}$. $p_{i,j}$ is either a valid pointer to some other cell of M, or is interpreted as $\perp$ (null). If $p_{i,j}$ is not a valid pointer to some other cell, we write "$p_{i,j} = \perp$".

Now, we define what we call a *valid pointer chain*. Assume that $t = \sqrt{n}$. For an input M to F, a sequence $((i_1, j_1), \ldots, (i_t, j_t))$ of indices in $[\sqrt{n}] \times [\sqrt{n}]$ is called a *valid* pointer chain if:

1. $b_{i_1,j_1} = 1$;
2. $b_{i_2,j_2} = \ldots = b_{i_t,j_t} = 0$;
3. $\forall k < i_1, p_{k,j_1} = \perp$;
4. for $\ell = 1, \ldots, t-1, p_{i_\ell,j_\ell} = (i_{\ell+1}, j_{\ell+1})$ and $p_{i_t,j_t} = \perp$;
5. $\{b_1, \ldots, b_t\} = \lfloor t \rfloor$;

F evaluates to 1 on M iff the following is true:

1. M contains a unique all 1's column j_1, i.e., there exists $j_1 \in [\sqrt{n}]$ such that $\forall i \in [\sqrt{n}]$, $b_{i,j_1} = 1$.

2. There exists a valid pointer chain $((i_1, j_1), \ldots, (i_t, j_t))$. This means that the column j_1 has a cell with non-null pointer entry. (i_1, j_1) is the cell on column j_1 with minimum row index whose pointer-entry is non-null. Starting from p_{i_1,j_1}, if we follow the successive pointers, the following conditions are satisfied: In each step except the last, the cell reached by following the pointer-entry of the cell in the previous step, contains a 0 as bit-entry and a non-null pointer as pointer-entry. In the last step, the cell contains a zero as bit-entry and a null pointer ($\perp$) as pointer-entry. Also, this pointer chain covers each column of M exactly once.

By a simple adversarial strategy, Göös et al. [4] showed that $D(F) = \widetilde{\Omega}(n)$. Our contribution is to show the following results.

▶ **Lemma 3.** *For the function F defined above, $R_0(F) = \widetilde{O}(n^{3/4})$.*

▶ **Lemma 4.** *For the function F defined above, $R_1(F) = \widetilde{O}(\sqrt{n})$.*

Clearly, Lemmas 3 and 4 imply Theorems 1 and 2 respectively.

2 Randomized One-sided Error Query Algorithm for F

We show that the randomized one-sided error query complexity of F is $\widetilde{O}(\sqrt{n})$. We first provide intuition for our one-sided error query algorithm for F before formally describing it.

Broad idea: Our algorithm errs on one side: on 0-inputs it always outputs 0 and on 1-inputs it outputs 1 with high probability.
The algorithm attempts to find a 1-certificate. If it fails to find a 1-certificate, it outputs 0. We show that on every 1-input, with high probability, the algorithm succeeds in finding a 1-certificate. The 1-certificate our algorithm looks for consists of:

1. A column j, all of whose bit-entries are 1's.
2. All null pointers of column j till its first non-null pointer-entry.
3. The pointer chain of length $\sqrt{n}$ that starts from the first non-null pointer entry, and in the next $\sqrt{n} - 1$ hops, visits all the other columns. The bit entries of all the other cells of the pointer chain than the one in this column are 0.

To find a 1-certificate, the algorithm tries to find columns with 0-cells on them, and adds those columns to a set of discarded columns that it maintains. To this end, a first natural attempt is to repeatedly sample a cell randomly from M, and if its bit-entry is 0, try to follow the pointer originating from that cell. Following the chain, each time we visit a cell with bit-entry 0, we can discard the column on which the cell lies. We can expect that, with high probability, after sampling $O(\sqrt{n})$ cells, we land up on some cell in the middle portion of the correct pointer chain that is contained in the 1-certificate (we call this the *principal chain*). Then if we follow that pointer we spend $O(\sqrt{n})$ queries, and eliminate a constant fraction of the existing columns.

The problem with this approach is possible existence of other long pointer chains, than the principal chain. It may be the case that we land up on one such chain, of $\Omega(\sqrt{n})$ length, which passes entirely through the columns that we have already discarded. Thus we end up spending $\Omega(\sqrt{n})$ queries, but can discard only one column (the one we began from).

To bypass this problem, we start by observing that the principal chain passes through every column, and hence in particular through every undiscarded column. Let N be the number of undiscarded column at some stage of the algorithm. Note that the length of the principal chain is $\sqrt{n}$. Therefore if we start to follow it from a randomly chosen cell on it,

Algorithm 1

```
1: procedure MilestoneTrace(M, C, i, j)
2:     Read b_{i,j};
3:     if b_{i,j} = 1 then return ;
4:     end if
5:     step:=0;
6:     discard:=1;
7:     current:=(i, j);
8:     seen:={j};
9:     while step ≤ 100√n · discard/|C| do
10:        read the pointer-entry of current;
11:        step ← step+1;
12:        current ← pointer-entry of current;
13:        if current is ⊥ then goto step21;
14:        end if
15:        read bit-entry of current;
16:        if current is on a column k in C \ seen and bit-entry of current is 0 then
17:            seen ← seen ∪{k};
18:            discard ← discard+1;
19:        end if
20:    end while
21:    C ← C \ seen;
22: end procedure
```

we are expected to see an undiscarded column in roughly another $\sqrt{n}/N$ hops. In view of this, we modify our algorithm as follows: while following a pointer chain, we check if on an average we are seeing one undiscarded column in every $O(\sqrt{n}/N)$ hops. If this check fails, we abandon following the pointer, sample another random cell from M, and continue. Our procedure MilestoneTrace does this pointer-traversal. We can prove that conditioned on the event that we land up on the principal chain, the above traversal algorithm enables us to eliminate a constant fraction of the existing undiscarded columns with high probability. We also show that spending $O(\sqrt{n}/N)$ queries for each column we eliminate is enough for us to get the desired query complexity bound.

After getting hold of the unique all 1's column, the final step is to check if all its bit-entries are indeed 1's, and if that can be completed into a full 1-certificate. That can clearly be done in $\widetilde{O}(\sqrt{n})$ queries. The VerifyColumn procedure does this.

2.1 The Algorithm

In this section we give the formal description and analysis of our one-sided error query algorithm for F: Algorithm 3. Algorithm 3 uses two procedures: VerifyColumn (see Algorithm 2) and MilestoneTrace (see Algorithm 1). As outlined in the previous section, VerifyColumn, given a column, checks if all its bit-entries are 1 and whether it can be completed into a 1-certificate. MilestoneTrace procedure implements the pointer traversal algorithm that we described in the preceding paragraph. We next describe the MilestoneTrace procedure in a little more detail. We recall from the last section that the algorithm discards columns in course of its execution. We denote the set of undiscarded columns by $\mathcal{C}$.

Algorithm 2

```
1: procedure VerifyColumn(M, k)
2:     Check if all the bit-entries of cells in the k-th column of M are 1; If not, output 0;
3:
4:     if All the pointer-entries of cells in the the k-th column of M are ⊥ then
5:         Output 0;
6:     end if
7:     if The pointer chain starting from the first non-null pointer in column k is valid then
8:         Output 1;
9:     else
10:        Output 0;
11:    end if
12: end procedure
```

MilestoneTrace procedure: The functions of the variables used are as follows:

1. *step*: Contains the number of pointer-entries queried so far. A bit query is always accompanied by a pointer query, unless the bit is 1 in which case the traversal stops. So upto logarithmic factor, the value in *step* gives us the number of bits queried.
2. *seen*: Set of columns that were undiscarded before the current run of MilestoneTrace, and that have so far been seen and marked for discarding.
3. *discard*: size of *seen*
4. *current*: Contains the indices of the cell currently being considered.

The condition in the *while* loop checks if the number of queries spent is not too much larger than $\frac{\sqrt{n}}{|\mathcal{C}|}$ at any point in time. The *if* condition in line 13 checks if the current pointer-entry is null. If it is null, $\mathcal{C}$ is updated, and control returns to Algorithm 3. The condition in line 13 checks if the pointer chain has reached its end.

To analyse Algorithm 3, we need to prove two statements about MilestoneTrace, which we now informally state. Assume that the algorithm is run on a 1-input.

1. Conditioned on the event that a cell (i, j) randomly chosen from the columns in $\mathcal{C}$ is on the principal chain, a call to MilestoneTrace$(M, \mathcal{C}, i, j)$ serves to eliminate a constant fraction of surviving columns with high probability.
2. For upper bounding the number of queries, it is enough to ensure that the average number of queries spent for each eliminated column is not too much larger than $\frac{\sqrt{n}}{|\mathcal{C}|}$. Note that $|\mathcal{C}|$ is the number of undiscarded columns during the start of the MilestoneTrace procedure.

In Section 2.2, we prove that Algorithm 3 makes $\widetilde{O}(\sqrt{n})$ queries on every input. In Section 2.3 we prove that Algorithm 3 succeeds with probability 1 on 0-inputs and with probability at least 2/3 on 1-inputs.

2.2 Query complexity of Algorithm 3

In this subsection we analyse the query complexity of Algorithm 3. We bound the total number of $b_{i,j}$'s and $p_{i,j}$'s read by the algorithm. Upto logarithmic factors, that is the total number of bits queried. For the rest of this subsection, one query will mean one query to a bit-entry or a pointer-entry of some cell.

Algorithm 3

```
1: C := set of columns in M.
2: for t = 1 to O(√n log n) do
3:     if |C| < 100 then
4:         goto step 10;
5:     end if
6:     Sample a column j from C uniformly at random;
7:     Sample i ∈ [√n] uniformly at random;
8:     MilestoneTrace(M, C, i, j);
9: end for
10: if |C| > 100 or |C| = 0 then
11:     Output 0;
12: else
13:     Read all columns in C;
14:     if There is a column k with all bit-entries equal to 1 then
15:         VerifyColumn(M, k);
16:     else
17:         Output 0;
18:     end if
19: end if
```

We first analyse the MilestoneTrace procedure. Recall that $\mathcal{C}$ denotes the set of undiscarded columns.

▶ **Lemma 5.** *Let i, j be such that $b_{i,j} = 0$. Let Q and D respectively be the number of queries made and number of columns discarded by a call to* MilestoneTrace$(M, \mathcal{C}, i, j)$*. Then,*

$$Q \leq D \cdot \frac{200\sqrt{n}}{|\mathcal{C}|} + 3$$

Proof. We note that the variable *step* contains the number of pointer queries made so far, and the variable *discard* maintains the number of columns marked so far for discarding. Every time the *while* loop is entered, $step \leq 100\sqrt{n} \cdot \frac{discard}{|\mathcal{C}|}$. In each iteration of the *while* loop, step goes up by 1. So at any point, $step \leq 100\sqrt{n} \cdot \frac{discard}{|\mathcal{C}|} + 1$. The lemma follows by observing that the total number of bit-entries queried is at most one more than total number of pointer-entries queried. ◀

We now use Lemma 5 to bound the total number of queries made by Algorithm 3.

▶ **Lemma 6.** *Algorithm 3 makes $\widetilde{O}(\sqrt{n})$ queries on each input.*

Proof. Whenever $b_{i,j} = 1$, MilestoneTrace$(M, \mathcal{C}, i, j)$ returns after reading $b_{i,j}$. So the total number of queries made by all calls to MilestoneTrace$(M, \mathcal{C}, i, j)$ on such inputs is $\widetilde{O}(\sqrt{n})$.
After leaving the *while* loop, the total number of queries required to read constantly many columns in $\mathcal{C}$ and to run VerifyColumn is $\widetilde{O}(\sqrt{n})$.
Since inside the *while* loop all the queries are made inside the MilestoneTrace procedure, it is enough to show that the total number of queries made by all calls to MilestoneTrace$(M, \mathcal{C}, i, j)$ on inputs for which $b_{i,j} = 0$ is $\widetilde{O}(\sqrt{n})$.
Let $t = \widetilde{O}(\sqrt{n})$ be the total number of calls to MilestoneTrace on such inputs, made

in the entire run of Algorithm 3. Let s_i be the value of $|\mathcal{C}|$ when the i-th call to MILESTONETRACE is made, and let s_{t+1} be the value of $|\mathcal{C}|$ after the execution of the t-th call to MILESTONETRACE completes . Let Δs_i and Δq_i respectively be the number of columns discarded and number of queries made in the i-th call to MILESTONETRACE. Since $\mathcal{C}$ shrinks only when $b_{i,j} = 0$, we have $\Delta s_i = s_i - s_{i+1}$ for $i = 1 \dots t$. Since $s_1 = \sqrt{n}$, we have that for $i = 2, \dots, t$, $s_i = \sqrt{n} - \sum_{j=1}^{i-1} \Delta s_j$.

From Lemma 5 we have $\Delta q_i \leq \Delta s_i \cdot \frac{200\sqrt{n}}{s_i} + 3$ for $i = 1, \dots, t$. Substituting $\sqrt{n} - \sum_{j=1}^{i-1} \Delta s_j$ for s_i when $i > 1$, and adding, we have,

$$\begin{aligned}
\sum_{i=1}^{t} \Delta q_i &\leq 200\sqrt{n} \cdot \sum_{i=1}^{t} \frac{\Delta s_i}{s_i} + 3t \\
&= 200\sqrt{n} \cdot \left(\frac{\Delta s_1}{\sqrt{n}} + \sum_{i=2}^{t} \frac{\Delta s_i}{\sqrt{n} - \sum_{j=1}^{i-1} \Delta s_j} \right) + \widetilde{O}(\sqrt{n}) \\
&\leq 200\sqrt{n} \cdot \left(\left(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n}-1} + \ldots + \frac{1}{\sqrt{n} - \Delta s_1 + 1} \right) + \right. \\
&\qquad \left(\frac{1}{\sqrt{n} - \Delta s_1} + \frac{1}{\sqrt{n} - \Delta s_1 - 1} + \ldots + \frac{1}{\sqrt{n} - \Delta s_1 - \Delta s_2 + 1} \right) + \\
&\qquad \ldots + \left(\frac{1}{\sqrt{n} - \sum_{j=1}^{t-1} \Delta s_j} + \frac{1}{\sqrt{n} - \sum_{j=1}^{t-1} \Delta s_j - 1} + \right. \\
&\qquad \left. \left. \ldots + \frac{1}{\sqrt{n} - \sum_{j=1}^{t-1} \Delta s_j - \Delta s_t + 1} \right) \right) \\
&\qquad + \widetilde{O}(\sqrt{n}) \\
&\leq O(\sqrt{n}) \cdot \left(\sum_{i=1}^{\sqrt{n}} \frac{1}{i} \right) + \widetilde{O}(\sqrt{n}) \\
&= O(\sqrt{n} \log n) + \widetilde{O}(\sqrt{n}) \\
&= \widetilde{O}(\sqrt{n}).
\end{aligned}$$

Hence proved. ◀

2.3 Success Probability of Algorithm 3

In this section we prove that Algorithm 3 outputs correct answer with probability 1 on 0-inputs and with probability at least 2/3 on 1-inputs. We start by a proving a probability statement (Lemma 7) that will help us in the analysis.

Let $x_1, \dots, x_\ell$ be non-negative real numbers and $\sum_{i=1}^{\ell} x_i = N$. We say that an index $I \in [\ell]$ is *bad* if there exists a non-negative integer $0 \leq D \leq N - I$ such that

$$\sum_{i=I}^{I+D} x_i > 100(D+1) \cdot \frac{N}{\ell}$$

We say that an index I is *good* if I is not bad.

▶ **Lemma 7.** *Let I be chosen uniformly at random from $[\ell]$. Then,*

$$\mathbb{P}[I \text{ is good}] > \frac{99}{100}$$

Proof. We show existence of a set $K = \{J_1, \cdots, J_t\}$ of disjoint sub-intervals of $[1, \ell]$ with integer end-points, having the following properties:
1. Every bad index is in some interval $J_i \in K$.
2. $\forall 1 \leq i \leq t, \sum_{j \in J_i} x_j > 100|J_i| \cdot \frac{N}{\ell}$.

It then follows that the number of bad indices is upper bounded by $\sum_{i \in [t]} |J_i|$ (by property 1). But $N \geq \sum_{i \in [t]} \sum_{j \in J_i} x_j > 100 \cdot \frac{N}{\ell} \sum_{i \in t} |J_i|$, which gives us that $\sum_{i \in t} |J_i| < \frac{\ell}{100}$. In the above chain of inequalities, the first inequality follows from the disjointness of J_i's and the second inequality follows from property 2.

Now we describe a greedy procedure to obtain such a set K of intervals. Let J be the smallest bad index. Then there exists a D such that $\sum_{i \in [J, J+D]} x_i > 100(D+1) \cdot \frac{N}{\ell}$. We include the interval $[J, J+D]$ in K. Then let J' be the smallest bad index greater than $J+D$. Then there exists a D' for which $\sum_{i \in [J', J'+D']} x_i > 100(D'+1) \cdot \frac{N}{\ell}$. We include $[J', J'+D']$ in K. We continue in this way till there is no bad index which is not already contained in some interval in K. It is easy to verify that the intervals in the set K thus formed are disjoint, and the set K satisfies properties 1 and 2. ◀

Let us begin by showing that algorithm 3 is correct with probability 1 on 0-inputs of F.

▶ **Lemma 8.** *If Procedure* VerifyColumn *outputs* 1 *on inputs M and k, then M is a* 1*-input of F.*

Proof. VerifyColumn outputs 1 only if the column k has all its bit-entries equal to 1, and if the pointer chain starting from the first non-null pointer entry is valid (recall the definition of a *valid pointer chain* from Section 1.1). From the definition of F, for such inputs F evaluates to 1. ◀

▶ **Corollary 9.** *Let M be a* 0*-input of of F. Then algorithm 3 outputs* 0 *with probability* 1*.*

Proof. The corollary follows by observing that if algorithm 3 returns 1, a call to VerifyColumn also returns 1, and hence from Lemma 8 the input is a 1-input of F. ◀

Let us now turn to 1-inputs of F. Let M be a 1-input of F, that we fix for the rest of this subsection. Without explicit mention, for the rest of the subsection we assume that Algorithm 3 is run on M. Since M is a 1-input, by the definition of F, there is a column C such that all its bit-entries are 1, and the pointer chain starting from the first non-null pointer-entry of C is valid. Call this pointer chain the *principal chain*. Let $(C = c_1, \ldots, c_{\sqrt{n}})$ be the order of columns of M in which the pointer chain crosses them. Let $(C = m_1, \ldots, m_{|\mathcal{C}|})$ be the order of the columns of $\mathcal{C}$ in which the pointer chain crosses them. Note that the column C always belongs to $\mathcal{C}$, as a column is discarded only if the bit-entry of some cell on it is 0. Define X_i to be the number of c_j's between m_i and m_{i+1}, including m_i, if $i < |\mathcal{C}|$, and the number of c_j's after m_i, including m_i, if $i = |\mathcal{C}|$. Clearly $\sum_{i=1}^{|\mathcal{C}|} X_i = \sqrt{n}$.

▶ **Lemma 10.** *Let (i, j) be a randomly chosen cell on the restriction of the principal chain to the columns in $\mathcal{C}$. Let $j = m_\ell \in \{m_1, \ldots, m_{|\mathcal{C}|}\}$. Let $|\mathcal{C}| = N \geq 100$. Then with probability at least $\frac{97}{100}$ over the choice of (i, j), a run of the procedure* MilestoneTrace *on inputs $M, \mathcal{C}, i, j$ shrinks the size of $\mathcal{C}$ to at most $\frac{99N}{100}$.*

Proof. By applying Lemma 7 on the sequence $(X_i)_{i=1}^{|\mathcal{C}|}$ described in the paragraph preceding this lemma, except with probability at least $1/100 + 1/100 + 1/|\mathcal{C}| \leq 3/100$, ℓ is a good index, $\ell < \frac{99N}{100}$ (i.e. the column j has at least $\frac{N}{100}$ columns of $\mathcal{C}$ ahead of it on the principal chain), and $j \neq C$. Since $j \neq C$, the bit-entry of the cell sampled is 0, and hence procedure MilestoneTrace does not return control in step 3. In the procedure MilestoneTrace, if *current* is on the principal chain, the condition in line 13 cannot be satisfied unless *current* is the last cell on the chain. Now, if the condition in the while loop is violated while *current* is on the principal chain, it implies that j is a bad index. Thus with probability at least $1 - 3/100 = 97/100$, the procedure does not terminate as long as all the $\frac{N}{100}$ columns ahead of j are not seen. Since all columns in $\mathcal{C}$ that are seen are discarded, we have the lemma. ◀

Now, let us bound the number of iterations of the *for* loop of algorithm 3 required to shrink $|\mathcal{C}|$ by a factor of $1/100$.

▶ **Lemma 11.** *Assume that at a stage of execution of algorithm 3 where the control is in the beginning of the* for *loop,* $|\mathcal{C}| = N$*. Then except with probability* $1/25$*, after* $10\sqrt{n}$ *iterations of the* for *loop,* $|\mathcal{C}|$ *will become at most* $99N/100$*.*

Proof. The probability that a cell on the principal chain is sampled in steps 6 and 7 is $\frac{1}{\sqrt{n}}$. So the probability that in none of the $10\sqrt{n}$ executions of steps 6 and 7, a cell on the principal chain is picked is $(1 - \frac{1}{\sqrt{n}})^{10\sqrt{n}} \leq \frac{1}{100}$. Conditioned on the event that a cell on the principal chain is sampled, from lemma 10, except with probability $3/100$, $|\mathcal{C}|$ reduces by a factor of $1/100$ in the following run of MilestoneTrace. Union bounding we have that except with probability $1/100 + 3/100 = 1/25$, after $10\sqrt{n}$ iterations of the *for* loop, $|\mathcal{C}| \leq 99N/100$. ◀

Let t be the minimum integer such that $\sqrt{n} \cdot \left(\frac{99}{100}\right)^t < 100$. Thus $t = O(\log n)$. For $i = 1, \ldots, t$, let the random variable Y_i be equal to the index of the first iteration of the *for* loop of Algorithm 3 after which $|\mathcal{C}| \leq \sqrt{n}.\left(\frac{99}{100}\right)^i$. Let $Z_1 = Y_1$ and for $i = 2, \ldots, t$ define $Z_i = Y_i - Y_{i-1}$. From Lemma 11, for each i we have $\mathbb{E}[Z_i] \leq 25 \times 10\sqrt{n} = O(\sqrt{n})$. By linearity of expectation, we have $\mathbb{E}[\sum_{i=1}^{t} Z_i] = O(\sqrt{n}\log n)$. By Markov's inequality, with probability at least $2/3$, $\sum_{i=1}^{t} Z_i = O(\sqrt{n}\log n)$. Thus, if we choose the constant hidden in the number of iterations of the *for* loop of Algorithm 3 large enough, then with probability at least $2/3$, $|\mathcal{C}|$ shrinks to less than 100. Then the VerifyColumn procedure reads all the columns in $\mathcal{C}$ and outputs the correct value of F. Thus we have proved the following Lemma.

▶ **Lemma 12.** *With probability at least* $2/3$*, algorithm 3 outputs* 1 *on a* 1*-input.*

Lemma 4 follows from Lemma 6, Corollary 9 and Lemma 12.

3 Randomized Zero-error Query Algorithm for F

We first present a randomized query algorithm which satisfies the following conditions: If the algorithm outputs 0 then the given input is a 0-input (the algorithm actually exhibits a 0-certificate) and if the given input is a 0-input, then the algorithm outputs 0 with high probability. This algorithm makes $\tilde{O}(n^{3/4})$ queries in worst case. For the randomized zero-error algorithm we run Algorithm 3 and this algorithm one after another. If Algorithm 3 outputs 1 then we stop and output 1. Else, if Algorithm 4 says 0, we stop and output

0. Otherwise, we repeat. By the standard argument of $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$ we get the randomized zero-error algorithm. Though the query complexity of Algorithm 3 is $\tilde{O}(\sqrt{n})$, we get the zero-error query complexity of F to be $\tilde{O}(n^{3/4})$ because of the query complexity of Algorithm 4.

Now we define *column covering* and *column span* which we will use next.

▶ **Definition 13.** For two columns C_i and C_j ($C_i(C_j)$ denotes the i-th (j-th) column) in input matrix M, we say C_j is covered by C_i if there is a cell (k, i) in C_i and a sequence $(\beta_1, \delta_1), \ldots, (\beta_t, \delta_t)$ of pairs from $[\sqrt{n}] \times [\sqrt{n}]$ such that:

1. $b_{k,i} = 0$,
2. $\delta_t = j$,
3. for all $\ell \in [t]$, $b_{\beta_\ell, \delta_\ell} = 0$ and
4. $p_{k,i} = (\beta_1, \delta_1)$ and for $\ell = 1, \ldots, t-1, p_{(\beta_\ell, \delta_\ell)} = (\beta_{\ell+1}, \delta_{\ell+1})$.
5. For $1 \leq k < \ell \leq t, \delta_k \neq \delta_\ell$ and for $1 \leq k \leq t, i \neq \delta_k$.

▶ **Definition 14.** For a column C, we define Span_C to be the subset of columns in M which consists of C and any column which is covered by C.

We first give an informal description of the algorithm and then we proceed to formally analyze the algorithm in Section 3.1. As mentioned before this is also a one-sided algorithm, i.e., it errs on one side but it errs on the different side than that of Algorithm 3. The 0-certificates it attempts to capture are as follows:

1. If each of the columns has a cell with bit-entry 0, then the function evaluates to 0. Those bit-entries form a 0-certificate. If there are many 0's in each column, The algorithms may capture such a certificate in the first phase (*sparsification*).
2. Two columns C_1 and C_2 in M such that $C_1 \notin \mathsf{Span}_{C_2}$ and $C_2 \notin \mathsf{Span}_{C_1}$. Existence of two such columns makes the existence of a valid pointer chain impossible. This is captured in the second phase of the algorithm.
3. Lastly, if there is a column all of whose bit-entries are 1, which does not have a valid pointer chain, then that is also a 0-certificate. The algorithm may capture such a certificate in the last phase.

The algorithm proceeds as follows: The main goal of the algorithm is to eliminate any column where it finds a 0 in any of its cells. First the algorithm filters out the columns with large number of 0's with high probability by random sampling. The algorithm probes $\widetilde{O}(n^{1/4})$ locations at random in each column and if it finds any 0 in any column, it eliminates that column. This step is called *sparsification*. After sparsification, we are guaranteed that all the columns have small number of 0's. Now the remaining columns can have either of the following two characteristics: First, a large number of the columns in existing column set have large span. This implies that if we choose a column randomly from the existing columns, the column will span a large number of columns (i.e., a constant fraction of existing columns) with high probability and we can eliminate all of them. The algorithm does this exactly in the **procedure A** of the second phase. The other case can be where most of the columns have small spans. We can show that if this is the case, then if we pick two random columns C_i and C_j from the set of existing columns, C_i will not lie in the span of C_j and vice-versa with high probability, certifying that F is 0. This case is taken care of in the **procedure B** of the second phase of the algorithm.

The algorithm runs **procedure A** and **procedure B** one after another for logarithmic number of steps. If at any point of the iteration, the algorithm finds two columns which

are not in span of each other, the algorithm outputs 0 and terminates. Otherwise, as the **procedure A** decreases the number of existing columns by a constant factor in each iteration, with logarithmic number of iteration, either we completely exhaust the column set, which is again a 0-certificate, or we are left with a single column. Then the algorithm checks the remaining column and the validity of the pointer chain if that column is an all 1's column and answers accordingly. This captures the third kind of 0-certificate as mentioned before.

In Algorithm 4, we set τ to be the least number such that $\sqrt{n} \cdot (\frac{99}{100})^{\tau} \leq 1$. Clearly $\tau = O(\log n)$. We also set α to an appropriate constant.

3.1 Analysis of Algorithm 4

Let's first look at the query complexity of the algorithm.

▶ **Lemma 15.** *The query complexity of Algorithm 4 is $\widetilde{O}(n^{3/4})$ in worst case.*

Proof. We count the number of bit-entries and pointer-entries of the input matrix the algorithm probes. Up to logarithmic factor, that is asymptotically same as the number of bits queried.
The first *for* loop runs for $\sqrt{n}$ iteration and in each iteration samples T cells from a column. So the number of probes of the first *for* loop is $O(\sqrt{n} \times T) = \widetilde{O}(n^{3/4})$.

In **procedure A**, the number of probes needed to scan the column and to trace pointer from the column is $\widetilde{O}(n^{3/4})$. In **procedure B**, the algorithm has to check the span of two columns, which takes $\widetilde{O}(n^{3/4})$ probes. The number of iterations of the *for* loop of line 9 is at most $\tau = O(\log n)$. Hence the total number of probes made inside the *for* loop is $\widetilde{O}(n^{3/4})$.

Lastly, VerifyColumn takes $O(\sqrt{n})$ probes. So the total number of probes is bounded by $\widetilde{O}(n^{3/4})$. Thus the claim follows. ◀

The first *for* loop, i.e., line 3 to 8 is called *sparsification*. We have the following guarantee after sparsification.

▶ **Lemma 16.** *After the sparsification, with probability at least* 99/100*, every column in $\mathcal{C}$ has at most $n^{1/4}$ cells with bit-entry* 0*.*

Proof. We will bound the probability that all the T probes in a column outputs 1 conditioned on the fact that the column has more than $n^{1/4}$ 0's. A single probe in such a column outputs 0 with probability at least $1/n^{1/4}$. Hence all the probes output 1 with probability $(1-1/n^{1/4})^T \leq \frac{1}{100\sqrt{n}}$. By union bound, this happens to some column in M with probability at most $1/100$. ◀

This implies that except with probability $1/100$, the *if* conditions of lines 20 and 32 are never satisfied.

▶ **Lemma 17.** *Either of the following is true in each iteration of the* for *loop of line 9:*
1. *for a random column $C \in \mathcal{C}$, $|\mathsf{Span}_C| > |\mathcal{C}|/100$ with probability at least* $1/100$*.*
2. *For two randomly picked columns C_i and C_j in $\mathcal{C}$, with probability at least* $24/25$*, $C_j \notin \mathsf{Span}_{C_i}$ and $C_i \notin \mathsf{Span}_{C_j}$.*

Proof. Suppose (1) does not hold. For two random columns C_i and C_j, Let L_i (L_j) be the event that $|\mathsf{Span}_{C_i}|$ ($|\mathsf{Span}_{C_j}|$)$| > |\mathcal{C}|/100$. Let $E_{i,j}$ ($E_{j,i}$) be the event that $C_j \in \mathsf{Span}_{C_i}$ ($C_i \in \mathsf{Span}_{C_j}$). Thus we have,

$$\mathbb{P}\{E_{i,j}\} = \mathbb{P}\{L_i\} \cdot \mathbb{P}\{E_{i,j}|L_i\} + \mathbb{P}\{\overline{L_i}\} \cdot \mathbb{P}\{E_{i,j}|\overline{L_i}\}$$

Algorithm 4

1: $\mathcal{C}$:= Set of columns in M;
2: τ := Least number such that $\sqrt{n} \cdot (\frac{99}{100})^{\tau} \leq 1$;
3: **for** each column C in $\mathcal{C}$ **do**
4: Sample $T = 10 \cdot n^{1/4} \log n$ cells uniformly at random;
5: **if** any bit-entry of any cell is 0 **then**
6: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C\}$;
7: **end if**
8: **end for**
9: **for** $t = 1$ to τ **do**
10: **if** $|\mathcal{C}| \leq 1$ **then**
11: goto step 40
12: **end if**
13: **repeat**
14: **procedure** A
15: Sample a column C from $\mathcal{C}$ uniformly at random;
16: Read all entries of all cells of C;
17: **if** All bit-entries are 1 **then**
18: VerifyColumn(M, C);
19: **end if**
20: **if** Number of 0 bit-entries in $C > n^{1/4}$ **then**
21: Output 1 and abort;
22: **end if**
23: For each cell on C with bit-entry 0, trace pointer and compute Span_C;
24: $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathsf{Span}_C$;
25: **end procedure**
26: **until** $\alpha \log \log n$ times
27: **procedure** B
28: Pick two columns C_1 and C_2 uniformly at random from $\mathcal{C}$;
29: **if** All bit-entries of C_1 (C_2) are 1 **then**
30: VerifyColumn(M, C_1) (VerifyColumn(M, C_2));
31: **end if**
32: **if** Number of 0 bit-entries in C_1 or $C_2 > n^{1/4}$ **then**
33: Output 1 and abort;
34: **end if**
35: **if** $C_2 \notin \mathsf{Span}_{C_1}$ and $C_1 \notin \mathsf{Span}_{C_2}$ **then**
36: Output 0 and abort;
37: **end if**
38: **end procedure**
39: **end for**
40: **if** $\mathcal{C} = \emptyset$ **then**
41: Output 0;
42: **end if**
43: **if** $|\mathcal{C}| = 1$ **then**
44: Let $\mathcal{C} = \{C\}$.
45: VerifyColumn(M, C);
46: **end if**
47: Output 1.

$$\leq \mathbb{P}\{L_i\} + \mathbb{P}\{E_{i,j}|\overline{L_i}\}$$
$$\leq \frac{1}{100} + \frac{1}{100} = \frac{1}{50}$$

Similarly $\mathbb{P}\{E_{j,i}\} \leq \frac{1}{50}$. By union bound, (2) is true; ◀

Now we are ready prove the correctness of the algorithm.

▶ **Lemma 18.** *Given a* 0*-input, Algorithm 4 outputs* 0 *with probability at least* 19/20 .

Proof. We first note that after the execution of *for* loop in line 3, except with probability at most 1/100 there is no column in $\mathcal{C}$ having more than $n^{1/4}$ cells with bit-entries 0.

If the algorithm finds a column all of whose bit-entries are 1, it gives correct output by a run of VerifyColumn.

Next, we note that if in any iteration of the *for* loop (line 9), condition (2) of Claim 17 is satisfied, then we find a 0-certificate (i.e. a pair of columns, none of which lies in the span of the other) with probability at least 24/25.

Finally, assume that for each iteration of the *for* loop, condition (2) is not satisfied. This implies that for each iteration of the loop, condition (1) is satisfied (From Claim 17). As we run **procedure A** $\alpha \log\log n$ times, with probability at least $1 - (\frac{99}{100})^{\alpha \log\log n} \geq 1 - \frac{1}{100\tau}$ (for appropriate setting of the constant α) we land up on a column whose span is at least $|\mathcal{C}|/100$ and hence we eliminate 1/100 fraction of columns in $\mathcal{C}$, in one of the iterations of the inner *repeat* loop (line 13). By union bound, the probability that there is even one bad *repeat* loop where we do not eliminate $|\mathcal{C}|/100$ columns, is at most 1/100. Thus the probability that after the execution of *for* loop is over, $|\mathcal{C}| > 1$, is at most 1/100. So, the total error probability is bounded by $1/100 + \max\{1/25, 1/100\} = 1/20$ from which the claim follows. ◀

▶ **Lemma 19.** *Given a* 1*-input, Algorithm 4 outputs 1 with probability 1.*

Proof. The proof of this claim is straight-forward. As mentioned before, Algorithm 4 outputs 0 only if it finds a 0-certificate. As there is no 0-certificate for a 1-input , the algorithm outputs 1. ◀

Lemma 3 follows by combining Lemma 4, Lemma 18, Lemma 16 and Lemma 19.

▶ Remark. It is observed [1] that if we consider a slight variant of the function F, where the input matrix is a $n^{2/3} \times n^{1/3}$ matrix instead of $\sqrt{n} \times \sqrt{n}$ and modify Algorithm 4 accordingly, we get a $\widetilde{O}(n^{2/3})$ algorithm. It is to be noted that the query complexity of Algorithm 3 (modified accordingly) worsens to $\widetilde{O}(n^{2/3})$ for this function. This shows that for a minor variant of the function F, our algorithm can show a better separation between deterministic and zero-error randomized query complexity. However, the modified function cannot show the widest separation between deterministic and bounded error randomized query complexity.

Acknowledgments. We thank Arkadev Chattopadhyay, Prahladh Harsha and Srikanth Srinivasan for useful discussions.

References

1 Scott Aaronson. A query complexity breakthrough. shtetl-optimized.
2 Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *CoRR*, abs/1506.04719, 2015.

3 Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 118–126, 1987.

4 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:50, 2015.

5 Juris Hartmanis and Lane A. Hemachandra. One-way functions, robustness, and the non-isomorphism of np-complete sets. In *Proceedings of the Second Annual Conference on Structure in Complexity Theory, Cornell University, Ithaca, New York, USA, June 16-19, 1987*, 1987.

6 Noam Nisan. CREW prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991.

7 Ronald L. Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theor. Comput. Sci.*, 3(3):371–384, 1976.

8 Michael E. Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 29–38, 1986.

9 Miklos Santha. On the monte carlo boolean decision tree complexity of read-once formulae. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 180–187, 1991.

10 Marc Snir. Lower bounds on probabilistic linear decision trees. *Theor. Comput. Sci.*, 38:69–82, 1985.

11 Gábor Tardos. Query complexity, or why is it difficult to seperate NP^a cap co NP^a from P^a by random oracles a? *Combinatorica*, 9(4):385–392, 1989.

Dimension, Pseudorandomness and Extraction of Pseudorandomness*

Manindra Agrawal[1], Diptarka Chakraborty[1], Debarati Das[2], and Satyadev Nandakumar[1]

1 **Indian Institute of Technology Kanpur, India**
{manindra, diptarka, satyadev}@cse.iitk.ac.in
2 **Charles University in Prague, Czech Republic**
debaratix710@gmail.com

Abstract

In this paper we propose a quantification of distributions on a set of strings, in terms of how close to pseudorandom a distribution is. The quantification is an adaptation of the theory of dimension of sets of infinite sequences introduced by Lutz. Adapting Hitchcock's work, we also show that the logarithmic loss incurred by a predictor on a distribution is quantitatively equivalent to the notion of *dimension* we define. Roughly, this captures the equivalence between pseudorandomness defined via indistinguishability and via unpredictability. Later we show some natural properties of our notion of dimension. We also do a comparative study among our proposed notion of dimension and two well known notions of computational analogue of entropy, namely HILL-type pseudo min-entropy and next-bit pseudo Shannon entropy.

Further, we apply our quantification to the following problem. If we know that the dimension of a distribution on the set of n-length strings is $s \in (0, 1]$, can we extract out $O(sn)$ pseudorandom bits out of the distribution? We show that to construct such extractor, one need at least $\Omega(\log n)$ bits of pure randomness. However, it is still open to do the same using $O(\log n)$ random bits. We show that deterministic extraction is possible in a special case - analogous to the bit-fixing sources introduced by Chor *et al.*, which we term *nonpseudorandom bit-fixing source*. We adapt the techniques of Gabizon, Raz and Shaltiel to construct a deterministic *pseudorandom extractor* for this source.

By the end, we make a little progress towards P vs. BPP problem by showing that existence of optimal stretching function that stretches $O(\log n)$ input bits to produce n output bits such that output distribution has dimension $s \in (0, 1]$, implies P=BPP.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Pseudorandomness, Dimension, Martingale, Unpredictability, Pseudoentropy, Pseudorandom Extractor, Hard Function

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.221

1 Introduction

Incorporating randomness in any feasible computation is one of the basic primitives in theoretical computer science. Fortunately, any efficient (polynomial time) randomized algorithm does not require pure random bits. What it actually needs is a source that *looks* random to it and this is where the notion of *pseudorandomness* [4, 32] comes into picture. Since

* Research supported in part by Research-I Foundation and the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 616787.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 221–235

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

its introduction, pseudorandomness has been fundamental to the domain of cryptography, complexity theory and computational learning theory. Pseudorandomness is mainly a computational approach to study the nature of randomness, and *computational indistinguishability* [10] played a pivotal role in this. Informally, a distribution is said to be pseudorandom if no efficient algorithm can distinguish it from the uniform distribution. Another way of looking at computational indistinguishability is via the notion of *unpredictability* of distributions, due to Yao [32]. Informally, a distribution is *unpredictable* if there is no efficient algorithm that, given a prefix of a string coming from that distribution, can guess the next bit with a significant success probability. This line of research naturally posed the question of constructing algorithms that can generate pseudorandom distributions, known as *pseudorandom generators*. Till now we know such constructions by assuming the existence of *one-way functions*. It is well known that constructibility of an *optimal pseudorandom generator* implies complete derandomization (i.e., P=BPP) and *exponential hardness assumption* on one-way function enables us to do that. However, Nisan and Wigderson [25] showed that the existence of an exponential *hard function*, which is a much weaker assumption, is also sufficient for this purpose. The assumption was further weakened in [18].

In order to characterize the class of random sources, information theoretic notion of *min-entropy* is normally used. A computational analogue of entropy was introduced by Yao [32] and was based on compression. Håstad, Impagliazzo, Levin and Luby [12] extended the definition of min-entropy in computational settings while giving the construction of a pseudorandom generator from any one-way function. This HILL-type *pseudoentropy* basically extends the definition of pseudorandomness syntactically. Relations among above two types of pseudoentropy was further studied in [3]. A more relaxed notion of pseudoentropy, known as *next-bit Shannon pseudoentropy*, was later introduced by Haitner, Reingold and Vadhan [11] in the context of an efficient construction of a pseudorandom generator from any one-way function. In a follow up work [31], the same notion was alternatively characterized by *KL-hardness*. So far it is not clear which of the above notions is the most appropriate or whether they are at all suitable to characterize distributions in terms of the degree of pseudorandomness in it.

In this paper, we first propose an alternative measure to quantify the amount of pseudorandomness present in a distribution. This measure is motivated by the ideas of *dimension* [23] and *logarithmic loss unpredictability* [15]. Lutz used the betting functions known as *gales* to characterize the *Hausdroff dimension* of sets of infinite sequences over a finite alphabet. The definition given by Lutz cannot be carried over directly, because here we consider the distributions over finite length strings instead of sets containing infinite length strings. To overcome this difficulty, we allow "non-uniform" gales and introduce a new probabilistic notion of *success* of a gale over a distribution. We use this to define the *dimension* of a distribution. In [15], Hitchcock showed that the definition of dimension given by Lutz is equivalent to logarithmic loss unpredictability. In this paper, we show that this result can be adapted to establish a quantitative equivalence between the notion of logarithmic loss unpredictability of a distribution and our proposed notion of dimension. Roughly, this captures the essence of equivalence between pseudorandomness defined via indistinguishability and via unpredictability [32]. We show some important properties of the notion of dimension of a distribution, which eventually makes this characterization much more powerful and flexible. We also do a comparative study between our notion of dimension and two known notions of pseudoentropy, namely HILL-type pseudo min-entropy and next-bit pseudo Shannon entropy. We show that the class of distributions with high dimension is a strict superset of the class of distributions having high HILL-type pseudo min-entropy. Whereas, there is a much closer relationship between dimension and next-bit pseudo Shannon entropy.

Once we have a quantification of pseudorandomness of a distribution, the next natural question is how to extract the pseudorandom part from a given distribution. The question is similar to the question of constructing *randomness extractors* which is an *efficient* algorithm that converts a realistic source to an *almost* ideal source of randomness. The term *randomness extractor* was first defined by Nisan and Zuckerman [26]. Unfortunately there is no such deterministic algorithm and to extract out almost all the randomness, extra $\Omega(\log n)$ pure random bits are always required [27, 28]. There is a long line of research on construction of extractors towards achieving this bound. For a comprehensive treatment on this topic, we refer the reader to excellent surveys by Nisan and Ta-Shma [24] and Shaltiel [29]. Finally, the desired bound was achieved up to some constant factor in [20].

Coming back to the computational analogue, it is natural to study the same question in the domain of pseudorandomness. Given a distribution with dimension s, the problem is to output $O(sn)$ many bits that are pseudorandom. A simple argument can show that deterministic pseudorandom extraction is not possible, but it is not at all clear that how many pure random bits are necessary to serve the purpose. In this paper, we show that we need to actually involve $\Omega(\log n)$ random bits to extract out all the pseudorandomness present in a distribution. However explicit construction of one such extractor with $O(\log n)$ random bits is not known. If it is known that the given distribution has high HILL-type pseudo min-entropy, then any randomness extractor will work [3]. Instead of HILL-type pseudoentropy, even if we have Yao-type pseudo min-entropy, then also some special kind of randomness extractor (namely with a "reconstruction procedure") could serve our purpose [3]. Unfortunately both of these notions of pseudoentropy can be very small for a distribution with very high dimension. Actually the same counterexample will work for both the cases. So it is interesting to come up with an pseudorandom extractor for a class of distributions having high dimension.

As a first step towards this goal, we consider a special kind of source which we call the *nonpseudorandom bit-fixing source*. It is similar to the well studied notion of *bit-fixing random source* introduced by Chor *et al.* [5], for which we know the construction of a deterministic randomness extractor due to [19] and [8]. In this paper, we show that the same construction yields a deterministic pseudorandom extractor for all nonpseudorandom bit-fixing sources having *polynomial-size support*.

In the concluding section, we make a little progress towards the question of P vs. BPP by showing that in order to prove P=BPP, it is sufficient to construct an algorithm that stretches $O(\log n)$ pure random bits to n bits such that the output distribution has a non-zero dimension (not necessarily pseudorandom). The idea is that using such stretching algorithm, we easily construct a hard function, which eventually gives us the most desired optimal pseudorandom generator.

Notations: In this paper, we consider the binary alphabet $\Sigma = \{0, 1\}$. We denote $Pr_{x \in_R D}[E]$ as $D[E]$, where E is an event and x is drawn randomly according to the distribution D. We use U_m to denote the uniform distribution on Σ^m. Given a string $x \in \Sigma^n$, $x[i]$ denote the i-th bit of x and $x[1, \ldots, i]$ denotes the first i bits of x. Now suppose $x \in \Sigma^n$ and $S = \{s_1, s_2, \ldots, s_k\} \subseteq \{1, 2, \ldots, n\}$, then by x_S, we denote the string $x[s_1]x[s_2] \ldots x[s_k]$.

2 Quantification of Pseudorandomness

In this section, we propose a quantification of pseudorandomness present in a distribution. We adapt the notion introduced by Lutz [23] of an s-gale to define a variant notion of success

of an s-gale against a distribution D on Σ^n. Throughout this paper, we will talk about non-uniform definitions. First, we consider the definition of pseudorandomness.

2.1 Pseudorandomness

We start by defining the notion of *indistinguishability* which we will use frequently in this paper.

▶ **Definition 1** (Indistinguishability)**.** A distribution D over Σ^n is (S, ϵ)-*indistinguishable* from another distribution D' over Σ^n (for $S \in \mathbb{N}, \epsilon > 0$) if for every circuit C of size at most S, $|D[C(x) = 1] - D'[C(x) = 1]| \leq \epsilon$.

Now we are ready to introduce the notion of pseudorandomness.

▶ **Definition 2** (Pseudorandomness)**.** For a distribution D over Σ^n and for any $S > n$,[1] $\epsilon > 0$,

1. (via computational indistinguishability) D is said to be (S, ϵ)-*pseudorandom* if D is (S, ϵ)-indistinguishable from U_n; or equivalently,
2. (via unpredictability [32]) D is said to be (S, ϵ)-pseudorandom if $D[C(x_1, \cdots, x_{i-1}) = x_i] \leq \frac{1}{2} + \frac{\epsilon}{n}$ for all circuits C of size at most $2S$ and for all $i \in [n]$.

2.2 Martingales, s-gales and predictors

Martingales are "fair" betting games which are used extensively in probability theory (see for example, [2]). Lutz introduced a generalized notion, that of an s-gale, to characterize Hausdorff dimension [22] and Athreya *et al.* used a similar notion to characterize packing dimension[1].

▶ **Definition 3** ([22])**.** Let $s \in [0, \infty)$. An *s-gale* is a function $d : \Sigma^* \to [0, \infty)$ such that $d(\lambda) = 1$ and $d(w) = 2^{-s}[d(w0) + d(w1)], \forall w \in \Sigma^*$. A *martingale* is a 1-gale.

The following proposition establishes a connection between s-gales and martingales.

▶ **Proposition 4** ([22])**.** *A function $d : \Sigma^* \to [0, \infty)$ is an s-gale if and only if the function $d' : \Sigma^* \to [0, \infty)$ defined as $d'(w) = 2^{(1-s)|w|}d(w)$ is a martingale.*

In order to adapt the notion of an s-gale to the study of pseudorandomness, we first relate it to the notion of predictors, which have been extensively used in the literature [31]. Given an initial finite segment of a string, a predictor specifies a probability distribution over Σ for the next symbol in the string.

▶ **Definition 5.** A function $\pi : \Sigma^* \times \Sigma \to [0, 1]$ is a *predictor* if for all $w \in \Sigma^*$, $\pi(w, 0) + \pi(w, 1) = 1$.

Note that the above definition of a predictor is not much different from the type of predictor used in Definition 2. If we have a predictor that given a prefix of a string outputs the next bit, then by invoking that predictor independently polynomially many times we can get an estimate on the probability of occurrence of 0 or 1 as the next bit and using Chernoff bound it can easily be shown that the estimation is correct up to some inverse exponential error. For the detailed equivalence, the reader may refer to [31]. In this paper,

[1] Throughout this paper, we consider $S > n$ so that the circuit can at least read the full input; however reader can feel free to take any $S \in \mathbb{N}$.

we only consider the martingales (or s-gales) and predictors that can be computed using non-uniform circuits and from now onwards we refer them just by martingales (or s-gales) and predictors. And by the size of a martingale (or an s-gale or a predictor), we refer the size of the circuit corresponding to that martingale (or s-gale or predictor).

2.3 Conversion Between s-Gale & Predictor

There is an equivalence between an s-gale and a predictor. An early reference to this is [6]. We follow the construction given in [15].

A predictor π induces an s-gale d_π for each $s \in [0,\infty)$ and is defined as follows: $d_\pi(\lambda) = 1$, $d_\pi(wa) = 2^s d_\pi(w)\pi(w,a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$; equivalently $d_\pi(w) = 2^{s|w|}\prod_{i=1}^{|w|}\pi(w[1\cdots i-1], w[i])$ for all $w \in \Sigma^*$.

Conversely, an s-gale d with $d(\lambda) = 1$ induces a predictor π_d defined as: if $d(w) \neq 0$, $\pi_d(w,a) = 2^{-s}\frac{d(wa)}{d(w)}$; otherwise, $\pi_d(w,a) = \frac{1}{2}$, for all $w \in \Sigma^*$ and $a \in \Sigma$.

Hitherto, s-gales have been used to study the dimension of sets of infinite sequences - for an extensive bibliography, see [13] and [14]. Although in this paper, we consider distributions on finite length strings, the conversion procedure between s-gale and predictor will be exactly same as described above.

2.4 Defining Dimension

▶ **Definition 6.** An s-gale $d : \Sigma^* \to [0,\infty)$ is said to *ϵ-succeed over a distribution D* on Σ^n if $D[d(w) \geq 2] > \frac{1}{2} + \epsilon$.

Note that the above definition of win of an s-gale is not arbitrary and reader may refer to the last portion of the proof of Theorem 13 to get some intuition behind this definition. The following lemma states the equivalence between the standard definition of pseudorandomness and the definition using martingale.

▶ **Lemma 7.** *There exists a constant $c' > 0$ such that for every $c > c'$ and for any $n \in \mathbb{N}$, if a distribution D over Σ^n is (S,ϵ)-pseudorandom then there is no martingale of size at most $(S-c)$ that ϵ-succeeds on D. Conversely, if there is no martingale of size at most $3S$ that $\frac{\epsilon}{n}$-succeeds on D, then D is (S,ϵ)-pseudorandom.*

The proof of the above lemma follows from the fact that the martingale that wins on D, can act as a distinguisher circuit and conversely, if D is not pseudorandom then we have a next bit predictor which can be used to construct a martingale that will win on D. The next definition gives a complete quantification of distributions in terms of dimension.

▶ **Definition 8** (Dimension)**.** The (S,ϵ)-dimension of a distribution D on Σ^n is defined as

$$\dim_{S,\epsilon}(D) = \min\{1, \inf\{s \in [0,\infty) \mid \exists s-\text{ gale } d \text{ of size at most } S \text{ which } \epsilon\text{-succeeds on } D\}\}.$$

Informally, if the dimension of a distribution is s, we say that it is *s-nonpseudorandom*.

3 Unpredictability and Dimension

It is customary to measure the performance of a predictor utilizing a *loss function* [16]. The loss function determines the penalty incurred by a predictor for erring in its prediction. Let the next bit be b and the probability induced by the predictor on it is p_b.

Commonly used loss functions include the *absolute loss function*, which penalizes the amount $1 - p_b$; and the *logarithmic loss function*, which penalizes $-\log(p_b)$. The latter,

which appears complicated at first glance, is intimately related to the concepts of Shannon Entropy and dimension. In this section, adapting the result of Hitchcock [15], we establish that there is an equivalence between the notion of dimension that we define in the previous section, and the logarithmic loss function defined on a predictor.

▶ **Definition 9.** The *logarithmic loss function* on $p \in [0,1]$ is defined to be $loss(p) = -\log p$.

Using this, we define the running loss that a predictor incurs while it predicts successive bits of a string in Σ^n, as the sum of the losses that the predictor makes on individual bits.

▶ **Definition 10.** Let $\pi : \Sigma^* \times \Sigma \to [0,1]$ be a predictor.

1. The *cumulative loss* of π on $w \in \Sigma^n$, denoted as $\mathrm{Loss}(\pi, w)$, is defined by $\mathrm{Loss}(\pi, w) = \sum_{i=1}^{n} loss(\pi(w[1 \dots i-1]),\ w[i])$.
2. The *loss rate* of π on $w \in \Sigma^n$ is $\mathrm{LossRate}(\pi, w) = \frac{\mathrm{Loss}(\pi,w)}{n}$.
3. The ϵ-*loss rate* of π over a distribution D on Σ^n is $\mathrm{LossRate}_\epsilon(\pi, D) = \inf t + \frac{1}{n}$, where t is any number in $[0,1]$ such that $D[\mathrm{LossRate}(\pi, w) \leq t] > \frac{1}{2} + \epsilon$.

Note that for a fixed $n \in \mathbb{N}$, any distribution on Σ^n has loss rate between $\frac{1}{n}$ and 1. The unpredictability of a distribution is defined as the infimum of the loss rate that any predictor has to incur on the distribution.

▶ **Definition 11.** The (S, ϵ)-*unpredictability* of a distribution D on Σ^n is

$$\mathrm{unpred}_{S,\epsilon}(D) = \min\{1, \inf\{\mathrm{LossRate}_\epsilon(\pi, D) \mid \pi \text{ is a predictor of size at most } S\}\}.$$

With this, we can prove that dimension can equivalently be defined using unpredictability.

▶ **Theorem 12.** *For any distribution D on Σ^n, if $\dim_{S,\epsilon}(D) \leq s$, then $\mathrm{unpred}_{S^2,\epsilon}(D) \leq s$. Conversely, if $\mathrm{unpred}_{S,\epsilon}(D) \leq s$, then $\dim_{S^2,\epsilon}(D) \leq s$.*

The proof of the above theorem is motivated from the proof of the equivalence between logarithmic loss unpredictability and dimension [15].

Till this point, we have given all the definitions parameterized by the circuit size S and bias term ϵ. However, we can naturally extend our definitions to *asymptotic* definitions where we consider S to be any polynomial in n and ϵ to be inverse of any polynomial in n. In that case, we will get exact equivalence between dimension and unpredictability.

4 Properties of Dimension

We now establish a few basic properties of our notion of dimension. We begin by exhibiting a distribution on Σ^n with dimension s, for any $s \in (0,1]$.

First, we observe that the dimension of any distribution D is the infimum of a non-empty subset of $[0,1]$ and hence the dimension of a distribution is well-defined.

Since it is clear that any distribution on Σ^n has a dimension, the following theorem establishes the fact that our definition yields a nontrivial quantification of the set of distributions.

▶ **Theorem 13.** *Let $s \in (0,1]$. Then for large enough n and any $S > n$, $\epsilon > 0$, there is a distribution D on Σ^n with (S, ϵ)-dimension s.*

Proof. Let us take a distribution $D := U_n$, i.e., uniform distribution on Σ^n. If $s = 1$, then by Lemma 7, D is a distribution with the required (S, ϵ)-dimension, for any $S > 0$ and $\epsilon > 0$.

Otherwise, assume that $s \in (0,1)$. To each string $x \in \Sigma^n$, we append $\lfloor \frac{n}{s} \rfloor - n$ many zeros, and denote the resulting string as x'. Let $D'(x') = D(x)$. For strings $y \in \Sigma^{\lfloor \frac{n}{s} \rfloor}$ which do not terminate in a sequence of $\lfloor \frac{n}{s} \rfloor - n$ many zeros, we set $D'(y) = 0$.

Let $\pi : \Sigma^* \times \Sigma \to [0,1]$ be the predictor which testifies that the (S^2, ϵ)-unpredictability of $D \leq 1$. Define the new predictor $\pi' : \Sigma^* \times \Sigma \to [0,1]$ by

$$\pi'(x,b) = \begin{cases} \pi(x,b) & \text{if} |x| < n, b = 0,1 \\ 1 & \text{if} |x| \geq n, b = 0 \\ 0 & \text{otherwise.} \end{cases}$$

For every $w \in \Sigma^{\lfloor \frac{n}{s} \rfloor}$ which is in the support of D' such that $\text{LossRate}(\pi, w[1 \dots n]) \leq (1 + \epsilon_1 - \frac{1}{n})$, for any $\epsilon_1 > 0$, we have that

$$\text{LossRate}(\pi', w) = \frac{\text{Loss}(\pi, w[1 \dots n])}{\lfloor \frac{n}{s} \rfloor} \leq \frac{(1 + \epsilon_1 - \frac{1}{n})n}{\lfloor \frac{n}{s} \rfloor} \leq (s + \epsilon' - \frac{1}{\lfloor \frac{n}{s} \rfloor}),$$

for some $\epsilon' > 0$. The last inequality holds for small enough s/n and this testifies that the (S^2, ϵ)-unpredictability (hence the (S^4, ϵ)-dimension) of the distribution D' is at most s.

Now, assume that (S^4, ϵ)-dimension of D' is less than s and for some ϵ_1, $0 < \epsilon_1 < s$, there exists a s'-gale ($s' = s - \epsilon_1$) d of size at most S^4 which ϵ-succeeds on D'. We show that this would imply that D is not uniform. Now consider a string $w \in \Sigma^{\lfloor \frac{n}{s} \rfloor}$, which is in the support of D'. For any $k \in n+1, \cdots, \lfloor \frac{n}{s} \rfloor$, $d(w[1 \dots k]) \leq 2^{s'} d(w[1 \dots k-1])$ and thus $d(w) \geq 2$ will imply that $d(w[1 \dots n]) \geq 2^{-s'(\lfloor \frac{n}{s} \rfloor - n)+1}$. Now consider the martingale d' corresponding to the s'-gale d. According to [22], we have $d'(w') = 2^{(1-s')|w'|} d(w')$, for any string $w' \in \Sigma^*$. Thus,

$$\begin{aligned} D'[d'(w[1 \dots n]) \geq 2] &\geq D'[d(w[1 \dots n]) \geq 2^{-s'(\lfloor \frac{n}{s} \rfloor - n)+1}] \\ &\geq D'[d(w) \geq 2] \\ &> \frac{1}{2} + \epsilon. \end{aligned}$$

Note that $D'[d'(w[1 \dots n]) \geq 2]$ is same as $D[d'(x) \geq 2]$ or in other words $U_n[d'(x) \geq 2]$, which contradicts the fact that by Markov Inequality, $U_n[d'(x) \geq 2] \leq \frac{1}{2}$ and this completes the proof. ◀

In subsequent sections, we will see how to extract pseudorandom parts from a convex combination of distributions. We will need a weaker version of the following theorem which establishes a relationship between the dimension of a convex combination of distributions in terms of the dimension of its constituent distributions.

▶ **Theorem 14.** *Let D_1 and D_2 be the distributions on Σ^n and $\delta \in [0,1]$. Suppose D is the convex combination of D_1 and D_2 defined by $D = \delta D_1 + (1-\delta) D_2$. Then for any $S > n$ and $\epsilon > 0$, $\dim_{S,\epsilon}(D) \geq \min\{\dim_{S,\epsilon}(D_1), \dim_{S,\epsilon}(D_2)\}$.*

Proof. The claim clearly holds when δ is either 0 or 1, so assume that $0 < \delta < 1$. Let $\dim_{S,\epsilon}(D_1) = s_1$, and $\dim_{S,\epsilon}(D_2) = s_2$.

For the contrary, let us assume that, $\dim_{S,\epsilon}(D) < \min\{s_1, s_2\}$. Now consider $s = \min\{s_1, s_2\} - \epsilon_1$, for some $\epsilon_1, 0 < \epsilon_1 < \min\{s_1, s_2\}$. Then there exists an s-gale d of size at most S such that $D[d(w) \geq 2] > \frac{1}{2} + \epsilon$.

Let the string w for which $d(w) \geq 2$ holds be w_i, $1 \leq i \leq k$ and the corresponding probabilities in D be $p(w_i), 1 \leq i \leq k$. Let $q(w_i)$ and $r(w_i)$, $1 \leq i \leq k$, be the corresponding probabilities in D_1 and D_2 respectively. So, $\sum_{i=1}^{k} p(w_i) > \frac{1}{2} + \epsilon$, where $p(w_i) = \delta q(w_i) + (1-\delta) r(w_i), 1 \leq i \leq k$. Now, since $\dim_{S,\epsilon}(D_2) = s_2$, we have that $r(w_1) + \cdots + r(w_k) \leq \frac{1}{2} + \epsilon$. Thus $q(w_1) + \cdots + q(w_k) > \frac{1}{2} + \epsilon$ implying $\dim_{S,\epsilon}(D_1) < s_1$, which is a contradiction. ◀

If we just concentrate on pseudorandom distributions, then by replacing s-gales with martingales in the proof of the above theorem, we will get the following lemma, which will be used in Section 6.1.

▶ **Lemma 15.** *Let D_1 and D_2 be the (S,ϵ)-pseudorandom distributions on Σ^n for any $S > n$, $\epsilon > 0$ and $\delta \in [0,1]$. Suppose there exists a distribution D which can be expressed as $D = \delta D_1 + (1-\delta)D_2$, then D is also (S,ϵ)-pseudorandom.*

However, it is easy to see that convex combinations of distributions may have larger dimension than any of its constituents. For example, let us consider a $n \in \mathbb{N}$ and take the distribution U_n. Now take two distributions on Σ^{n+1}, namely, D_1 produced by the 0-dilution (padding each string with a 0 at the end) of U_n and D_2 produced by the 1-dilution (padding each string with a 1 at the end) of U_n. Then $D = 0.5D_1 + 0.5D_2$ is nothing but U_{n+1} and has dimension which exceeds the dimensions of D_1 and D_2 by $\frac{1}{n}$.

▶ **Theorem 16.** *Let D, D_1 and D_2 be the distributions on Σ^n, and consider $S > n$, $\epsilon > 0$ and $\delta \in [0,1]$. Suppose further that $\dim_{S,\epsilon}(D_1) = s_1$. Now if $D = (1-\delta)D_1 + \delta D_2$, then $\dim_{S,(\epsilon+\delta)}(D) \geq s_1$.*[2]

The proof of the above theorem is similar to that of Theorem 14. If we follow the proof of Theorem 16 with martingale instead of s-gale, we get the following weaker version of the above theorem, which we will require in the construction of deterministic extractor for a special kind of sources in Section 6.1.

▶ **Lemma 17.** *Let D, D_1 and D_2 be the distributions on Σ^n, and consider $S > n$, $\epsilon > 0$ and $\delta \in [0,1]$. If D_1 is (S,ϵ)-pseudorandom and $D = (1-\delta)D_1 + \delta D_2$, then D is $(S,\epsilon+\delta)$-pseudorandom as well.*

The following theorem shows that in order for a distribution to have dimension less than 1, it is not sufficient to have a few positions where we can successfully predict - it is necessary that these positions occur often.

▶ **Theorem 18.** *For large enough n and for any $S > n$ and $\epsilon > 0$, there is a distribution D_n on Σ^n such that $\dim_{S,\epsilon}(D_n) = 1$, but is not (S,ϵ)-pseudorandom.*

5 Pseudoentropy and Dimension

In this section we study the relation between our notion of dimension and different variants of computational or pseudo (min/Shannon) entropy.

5.1 High HILL-type pseudo min-entropy implies high dimension

For a distribution D, *min-entropy* of D is defined as $H_\infty(D) = \min_w\{\log(1/D[w])\}$. We start with the standard definition of computational min-entropy, as given by [12].

▶ **Definition 19** (HILL-type pseudo min-entropy)**.** A distribution D on Σ^n has (S,ϵ)-*HILL-type pseudo min-entropy* (or simply (S,ϵ)-pseudo min-entropy) at least k, denoted as $H_\infty^{HILL,S,\epsilon} \geq k$ if the there exists a distribution D' such that

1. $H_\infty(D') \geq k$, and
2. D' is (S,ϵ)-indistinguishable from the distribution D.

[2] Note that bias term in the dimension of D_1 depends on δ.

Several other definitions of pseudo min-entropy (metric-type, Yao-type or compression type) are there in the literature. We refer the reader to [3] for a comprehensive treatment on different definitions and the connections between them. In the remaining potion of this subsection, we focus only on HILL-type pseudo min-entropy. Now we state the main result of this subsection.

▶ **Theorem 20.** *There exists a constant $c' > 0$ such that for any $c > c'$, for every distribution D on Σ^n and for any $S > n$, $\epsilon > 0$, if $H_\infty^{HILL,(S+c),\epsilon}(D) \geq sn$, then $\dim_{S,\epsilon}(D) \geq s$*

Proof. The theorem is a consequence of the following claim.

▶ **Claim 21.** *For every distribution X on Σ^n, if $H_\infty(X) = k$ then $\dim_{S,\epsilon}(X) \geq k/n$, for any values of S and $\epsilon > 0$.*

Now observe that if a distribution D is $(S + c, \epsilon)$-indistinguishable from another distribution D', then $\dim_{S,\epsilon}(D) = \dim_{S,\epsilon}(D')$ as otherwise the s-gale which ϵ-succeeds over exactly one of them, acts as a distinguishing circuit. This fact along with Claim 21 completes the proof. ◀

It only remains to establish Claim 21.

Proof of Claim 21. Let us first take $s = k/n$. Now for the sake of contradiction, let us assume that there exists an s-gale d that ϵ-succeeds over X, i.e., $X[d(w) \geq 2] > \frac{1}{2} + \epsilon$. Now consider the set $S := \{w | d(w) \geq 2\}$. As $H_\infty(X) = k$, $|S| > 2^{sn-1} + 2^{sn}.\epsilon$. By taking the corresponding martingale d' according to the Proposition 4, we have that for any $w \in S$, $d'(w) \geq 2^{(1-s)n+1}$ and as a consequence, $U_n[d'(w) \geq 2^{(1-s)n+1}] > 2^{sn-n-1} + 2^{sn-n}.\epsilon$, which contradicts the fact that by Markov inequality, $U_n[d'(w) \geq 2^{(1-s)n+1}] \leq 2^{sn-n-1}$. ◀

The converse direction of the statement of Theorem 20 is also true if the distribution under consideration is pseudorandom. If the converse is true then we can apply any randomness extractor to get pseudorandom distribution from any distribution having high dimension [3]. However, we should always be careful about the circuit size with respect to which we call the output distribution pseudorandom. Unfortunately, in general the converse is not true.

Counterexample for the converse: Suppose *one-way functions* exist, then it is well-known that we can construct a *pseudorandom generator* $G : \Sigma^l \to \Sigma^m$ such that m is any polynomial in l, say $m = l^3$. For the definitions of one-way function, pseudorandom generator and the construction of pseudorandom generator with polynomial stretch from any one-way function, interested reader may refer to [9, 12, 31]. Now consider the distribution $D := (G(U_l), U_l)$. For large enough l, using the argument similar to the proof of Theorem 13, it can easily be shown that the distribution D has dimension almost 1 as the distribution on the first m bits are pseudorandom, but pseudo min-entropy is not larger than l.

5.2 Equivalence between dimension and next-bit pseudo Shannon entropy

We will use standard notions and notations of information theory (e.g., Shannon entropy, KL divergence) without defining them. Readers may refer to a book by Cover and Thomas [7] for the definitions.

In the last subsection, we have talked about pseudo min-entropy. In similar fashion, one can also define *pseudo Shannon entropy* and a natural generalization of it is *conditional pseudo Shannon entropy* [17, 11, 31].

▶ **Definition 22** (Conditional pseudo Shannon entropy)**.** Suppose Y is a random variable jointly distributed with X. Y is said to have (S, ϵ)-*conditional pseudo Shannon entropy* at least k given X if there exists a distribution Z jointly distributed with X such that
1. $H(Z|X) \geq k$, and
2. (X, Y) and (X, Z) are (S, ϵ)-indistinguishable.

The following is the variant of pseudoentropy that we are looking for in this subsection and was introduced by Haitner *et al.* [11].

▶ **Definition 23** (Next-bit pseudo Shannon entropy)**.** A random variable $X = (X_1, X_2, \cdots, X_n)$ taking values in Σ^n has (S, ϵ)-*next-bit pseudo Shannon entropy* at least k, denoted as $H^{next,S,\epsilon}(X) \geq k$ if there exist random variables $(Y_1, Y_2, \cdots, Y_n)$ such that
1. $\sum_i H(Y_i|X_1, \cdots, X_{i-1}) \geq k$, and
2. for all $1 \leq i \leq n$, $(X_1, \cdots, X_{i-1}, X_i)$ and $(X_1, \cdots, X_{i-1}, Y_i)$ are (S, ϵ)-indistinguishable.

Later, Vadhan and Zheng [31] provided an alternative characterization of conditional pseudo Shannon entropy by showing an equivalence between it and *KL-hardness* (defined below). We use this alternative characterization extensively for our purpose.

▶ **Definition 24** (KL-hardness)**.** Suppose (X, Y) is a $\Sigma^n \times \Sigma$-valued random variable and π be any predictor. Then π is said to be a δ-*KL-predictor* of Y given X if $\mathbf{KL}(X, Y \| X, C_\pi) \leq \delta$ where $C_\pi(y|x) = \pi(x, y)$ for all $x \in \Sigma^n$ and $y \in \Sigma$.

Moreover, Y is said to be (S, δ)-*KL-hard* given X if there is no predictor π of size at most S that is a δ-KL-predictor of Y given X.

The following theorem provides the equivalence among KL-hardness and conditional pseudo Shannon entropy of a distribution.

▶ **Theorem 25** ([31])**.** *For a $\Sigma^n \times \Sigma$-valued random variable (X, Y) and for any $\delta > 0$, $\epsilon > 0$,*
1. *If Y is (S, δ)-KL-hard given X, then for every $\epsilon > 0$, Y has (S', ϵ)-conditional pseudo Shannon entropy at least $H(Y|X) + \delta - \epsilon$, where $S' = S^{\Omega(1)}/poly(n, 1/\epsilon)$.*
2. *Conversely, if Y has (S, ϵ)-conditional pseudo Shannon entropy at least $H(Y|X) + \delta$, then for every $\sigma > 0$, Y is (S', δ')-KL-hard given X, where $S' = \min\{S^{\Omega(1)}/poly \log(1/\sigma), \Omega(\sigma/\epsilon)\}$ and $\delta' = \delta - \sigma$.*

Now we are ready to state the main theorem of this subsection which conveys the fact that the distributions with high dimensions also have high next-bit pseudo Shannon entropy.

▶ **Theorem 26.** *For any $\epsilon' > 0$, there exists a $n' \in \mathbb{N}$ such that for any $n \geq n'$ and $S > n$, $\epsilon > 0$, for every distribution D on Σ^n, if $\dim_{S,\epsilon}(D) > \frac{2s}{1-2\epsilon} + \epsilon'$, then $H^{next,S',\epsilon}(D) > sn$, where $S' = S^{\Omega(1)}/poly(n)$.*

To prove the above theorem, we first break D with dimension greater than $\frac{2s}{1-2\epsilon} + \epsilon'$ into 1-bit blocks, i.e., $D = (D_1, D_2, \cdots, D_n)$ and then by applying Item 1 of Theorem 25, we argue that next-bit pseudoentropy is at most sn implies unpredictability is at most $\frac{2s}{1-2\epsilon} + \epsilon'$ and thus get a contradiction.

On the contrary, for the other direction, the following weaker version can easily be proven.

▶ **Theorem 27.** *For any $\epsilon' > 0$, there exists a $n' \in \mathbb{N}$ such that for any $n \geq n'$ and $S > n$, $\epsilon > 0$, for every distribution D on Σ^n, if $H^{next,S,\epsilon}(D) > sn$, then $\dim_{S',\epsilon}(D) > s - \frac{1}{2} - \epsilon''$, where $S' = \min\{S^{\Omega(1)}/poly \log(1/\epsilon'), \Omega(\epsilon'/\sqrt{\epsilon})\}$ and $\epsilon'' = \epsilon' - \epsilon$.*

Technique used in the proof of the above theorem has a similar essence as of Theorem 26. The above two theorems can easily be extended to the asymptotic world in a natural way.

6 Pseudorandom Extractors & Lower Bound

We now introduce the notion of *pseudorandom extractor* similar to the notion of randomness extractor. Intuitively, a *randomness extractor* is a function that outputs almost random (statistically close to uniform) bits from weakly random sources, which need not be close to the uniformly random source. Two distributions X and Y on a set Λ are said to be $\epsilon - close$ (statistically close) if $\max_{S\subseteq\Lambda}\{|Pr[X \in S] - Pr[Y \in S]|\} \leq \epsilon$ or equivalently $\frac{1}{2}\sum_{x\in\Lambda}|Pr[X = x] - Pr[Y = x]| \leq \epsilon$.

▶ **Definition 28** (Deterministic Randomness Extractor)**.** A function $E : \Sigma^n \to \Sigma^m$ is said to be a deterministic ϵ-extractor for a class of distributions $\mathcal{C}$ if for every distribution X on n-bit strings in $\mathcal{C}$, the distribution $E(X)$ is ϵ-close to U_m.

Likewise, a seeded ϵ-extractor is defined and the only difference is that now it takes a d-bit string chosen according to an uniform distribution, as an extra input. Before going further, we mention that for ease of presentation, now onwards we will only talk about asymptotic versions of the definitions and results derived so far related to pseudorandomness and dimension. We now define the notion of a pseudorandom extractor, the purpose of which is to extract out pseudorandom distribution from a given distribution.

▶ **Definition 29** (Pseudorandom Extractor)**.** A function $E : \Sigma^n \to \Sigma^m$ is said to be a *deterministic pseudorandom extractor* for a class of distributions $\mathcal{C}$ if for every distribution X on n-bit strings in $\mathcal{C}$, $E(X)$ is pseudorandom.

A function $E : \Sigma^n \times \Sigma^d \to \Sigma^m$ is said to be a *seeded pseudorandom extractor* for a class of distributions $\mathcal{C}$ if for every distribution X on n-bit strings in $\mathcal{C}$, $E(X, U_d)$ is pseudorandom.

In this section, we will concentrate on the class of distributions having dimension at least s. It is clear from the results stated in Section 5.1 that this class of distribution is a strict superset of the class of distributions with HILL-type pseudo min-entropy at least sn, for which any randomness extractor will act as a pseudorandom extractor [3]. Thus it is natural to ask the following.

▶ **Question 1.** *For any $s \in (0, 1]$, does there exist a deterministic/seeded pseudorandom extractor for the class of distributions on Σ^n having dimension at least s?*

Just like the the case of randomness extraction, one can easily argue that deterministic pseudorandom extraction is not possible[3]. Now the most common question comes next is that what the lower bound on the seed length will be. We answer to this question in the following theorem.

▶ **Theorem 30.** *Suppose for any $s \in (0, 1]$, $E : \Sigma^n \times \Sigma^d \to \Sigma^m$ be a seeded pseudorandom extractor for the class of distributions on Σ^n having dimension at least s and for some $\delta > 0$, $m = (sn)^\delta$. Then $d = \Omega(\log n)$.*

Proof. For the sake of contradiction, let us assume that $d = o(\log n)$. Now by doing a walk according to the output distribution on an odd-length cycle, we achieve the following claim.

▶ **Claim 31.** *There is a deterministic $\frac{1}{\sqrt[4]{m}}$-extractor $E' : \Sigma^m \to \Sigma^{\frac{\log m}{4}}$ for all pseudorandom distributions on Σ^m.*

[3] Suppose $E : \Sigma^n \to \Sigma$ is a deterministic pseudorandom extractor, then there exists $x \in \Sigma$ such that $|E^{-1}(x)| \geq 2^{n-1}$. Thus E is not a pseudorandom extractor for a source D that is a uniform distribution on $E^{-1}(x)$ and by Claim 21, $\dim(D) \geq (1 - 1/n)$.

Now construct the following function $Ext : \Sigma^n \times \Sigma^d \to \Sigma^{c \log n}$ for some constant $c > 0$ such that $Ext(x, y) = E'(E(x, y))$ for all $x \in \Sigma^n, y \in \Sigma^d$. The function Ext is a seeded $\frac{1}{(sn)^{\delta/4}}$-extractor with $d = o(\log n)$, but it is well known due to [28](Theorem 1.9) that any such randomness extractor must satisfy $d = \Omega(\log n)$ and hence we get a contradiction. ◀

However, the question on constructing an *explicit* or polynomial time computable seeded pseudorandom extractor with seed length $O(\log n)$ is still open and next, we formally pose this question.

▶ **Question 2.** *For any $s \in (0, 1]$, can one construct a seeded pseudorandom extractor $E : \Sigma^n \times \Sigma^d \to \Sigma^m$ in polynomial time, for the class of distributions on Σ^n having dimension at least s such that $m = (sn)^\delta$ for some $\delta > 0$ and $d = O(\log n)$?*

In the next part of this section, we will see a special type of nonpseudorandom source and give an explicit construction of deterministic pseudorandom extractor for that particular type of source. Before proceeding further, we want to mention that it is also very interesting to consider *nonpseudorandom distributions samplable by poly-size circuits*, which is a natural extension of another special type of source called samplable source studied in [30]. By following the argument in [30], we can observe that the existence of deterministic pseudorandom extractor implies separation between deterministic complexity classes and non-uniform circuit classes which is not known so far. Nevertheless, it is still natural to ask the question of constructing explicit extractor using $O(\log n)$ amount of extra randomness for this special kind of source. We do not know any such result so far, but in Section 7 we will see that if some distribution is samplable using very few ($O(\log n)$) random bits, then it is possible to extract out all the pseudorandom bits using extra $O(\log n)$ random bits.

6.1 Deterministic Pseudorandom Extractor for Nonpseudorandom Bit-fixing Sources

In Section 4 while proving Theorem 13, we have introduced a special type of nonpseudorandom distribution which looks similar to the (n, k)-*bit-fixing source* defined as a distribution X over Σ^n such that there exists a subset $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ where all the bits at the indices of I are independent and uniformly chosen and rest of the bits are completely fixed. This distribution was introduced by Chor *et al.*[5]. Now we define an analogous notion for the class of nonpseudorandom distributions, which we term *nonpseudorandom bit-fixing sources.*

▶ **Definition 32** (Nonpseudorandom Bit-fixing Source)**.** Let $s \in (0, 1)$. For sufficiently large n and $\epsilon > 0$, a distribution D_n over Σ^n with dimension s is an (n, s, ϵ)-*nonpseudorandom bit-fixing source* if there exists a subset $I = \{i_1, \ldots, i_{\lceil sn \rceil}\} \subseteq \{1, \ldots, n\}$ such that all the bits at the indices of I come from an ϵ-pseudorandom distribution and rest of the bits are fixed.

We devote the rest of the section to achieve an affirmative answer to the question of constructing deterministic pseudorandom extractor for the nonpseudorandom bit-fixing sources. For this purpose, we show that a careful analysis of the technique used in the construction of the deterministic randomness extractor for bit-fixing random sources by Gabizon, Raz and Shaltiel [8] will lead us to the desired deterministic pseudorandom extractor.

▶ **Theorem 33.** *There exists a constant $c > 0$ such that for any $s \in (0, 1]$ and for large enough n, $0 < \epsilon < \frac{1}{\sqrt{n}}$, there is an explicit deterministic pseudorandom extractor $E : \Sigma^n \to \Sigma^m$ for all (n, s, ϵ)-nonpseudorandom bit-fixing sources having polynomial-size support, where $m = (sn)^{\Omega(1)}$.*

We first extract $O(\log sn)$ amount of almost random bits and then use the same as seed in the seeded extractor. To use the seeded extractor, we modify the source such that it becomes independent of the random bits extracted.

7 Approaching Towards P=BPP

We now show that if there is an exponential time computable algorithm $G : \Sigma^{O(\log n)} \to \Sigma^n$ where the output distribution has dimension s ($s > 0$), then this will imply P=BPP. We refer to this algorithm G as *optimal nonpseudorandom generator*. To prove the main result of this section, we use the following theorem proved by Impagliazzo and Wigderson.

▶ **Theorem 34** ([18]). *Suppose there is a language L in* EXP *and $\exists\delta > 0$ such that L on inputs of length n cannot be solved by circuits of size at most $2^{\delta n}$. Then there exists a language L' in* EXP *and $\exists\delta' > 0$ such that L' on inputs of length n is $(2^{\delta' n}, 1/2^{\delta' n})$-hard and as a consequence optimal pseudorandom generator exists.*

Now we use the above theorem in the proof of the following result.

▶ **Theorem 35.** *Consider any $s \in (0, 1]$ and $c > 0$. If there exists an algorithm $G_n : \Sigma^{c \log n} \to \Sigma^n$ computable in $2^{O(\log n)}$ such that for sufficiently large n, $\dim(G_n(U_{c \log n})) \geq s$, then* P=BPP*.*

Proof. Suppose $X := G_n(U_{c \log n})$. If $\dim(X) = s > 0$, then there must be a subset of indices $S \subseteq \{1, 2, \cdots, n\}$ such that $|S| = \log n$ and for any $i \in S$, loss incurred by any polynomial size predictor at i-th bit position is non-zero or in other words, for any poly-size circuit C, $X[C(x_1, \cdots, x_{i-1}) = x_i] < 1$. Otherwise according to Theorem 12 and by the argument used in the proof of Theorem 18, one can show that $\dim(X) = 0$, for large enough n. Suppose S contains first $\log n$ many such indices. Also assume that $S = \{i_1, i_2, \cdots, i_{\log n}\}$ and $i_1 < i_2 < \cdots < i_{\log n}$. Now we define two languages L_0 and L_1 as follows: for $j = 0, 1$, $L_j := \{y \in \Sigma^{\log n - 1} | \exists x \in \Sigma^n$ in the support of G_n and $x_S = jy\}$.

First of all, note that as $i_1 \in S$, none of L_0 and L_1 is a constant function. Now clearly either L_0 or L_1 is the language that satisfies all the conditions of Theorem 34 [18]. Otherwise, there exists a predictor circuit of size at most $2^{\delta \log n}$, for some $\delta > 0$, i.e., polynomial in n, by which we can predict $i_{\log n}$-th bit position or loss incurred by that predictor at $i_{\log n}$th bit position will be zero implying $i_{\log n} \notin S$ which is a contradiction. Thus either L_0 or L_1 can be used to construct an *optimal pseudorandom generator* and which eventually implies P=BPP. ◀

Acknowledgements. The authors thank Somenath Biswas for helpful discussions and comments. The second author also thanks Andrej Bogdanov for suggesting the study of the notion of pseudoentropy.

References

1 K. B. Athreya, J. M. Hitchcock, J. H. Lutz, and E. Mayordomo. Effective strong dimension, algorithmic information, and computational complexity. *SIAM Journal on Computing*, 37:671–705, 2007.

2 K. B. Athreya and S. N. Lahiri. *Measure Theory and Probability Theory*. Springer Verlag, 2006.

3 Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *RANDOM-APPROX*, volume 2764 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2003.

4 Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, November 1984.

5 Benny Chor, Oded Goldreich, Johan Håstad, Joel Freidmann, Steven Rudich, and Roman Smolensky. The bit extraction problem or t-resilient functions, 1985.

6 T. Cover. Universal gambling schemes and the complexity measures of Kolmogorov and Chaitin. Technical Report 12, Stanford University Department of Statistics, October 1974.

7 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

8 Ariel Gabizon, Ran Raz, and Ronen Shaltiel. Deterministic extractors for bit-fixing sources by obtaining an independent seed, 2005.

9 Oded Goldreich. *The Foundations of Cryptography – Volume 1, Basic Techniques*. Cambridge University Press, 2001.

10 Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

11 Iftach Haitner, Omer Reingold, and Salil P. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 437–446, 2010.

12 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

13 J. M. Hitchcock. Effective Fractal Dimension Bibliography, http://www.cs.uwyo.edu/~jhitchco/bib/dim.shtml (current April, 2011).

14 J. M. Hitchcock. Resource Bounded Measure – Bibliography, http://www.cs.uwyo.edu/~jhitchco/bib/rbm.shtml (current April, 2011).

15 J. M. Hitchcock. Fractal dimension and logarithmic loss unpredictability. *Theoretical Computer Science*, 304(1–3):431–441, 2003.

16 John M. Hitchcock. Fractal dimension and logarithmic loss unpredictability, 2004.

17 Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 2007.

18 Russell Impagliazzo and Avi Wigderson. P=BPP unless E has sub-exponential circuits: Derandomizing the xor lemma (preliminary version). In *In Proceedings of the 29th STOC*, pages 220–229. ACM Press, 1996.

19 Jesse Kamp and David Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. In *In Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 92–101, 2003.

20 Chi-Jen Lu, Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Extractors: optimal up to constant factors. In Lawrence L. Larmore and Michel X. Goemans, editors, *STOC*, pages 602–611. ACM, 2003.

21 J. H. Lutz. Gales and the constructive dimension of individual sequences. In *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming*, pages 902–913, 2000. Revised as [22].

22 J. H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*, 187:49–79, 2003. Preliminary version appeared as [21].

23 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003.

24 Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *J. Comput. Syst. Sci.*, 58(1):148–173, 1999.

25 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

26 Noam Nisan and David Zuckerman. More deterministic simulation in logspace. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 235–244. ACM, 1993.

27 Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52:43–52, 1996.

28 Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13:2000, 2000.

29 Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

30 Luca Trevisan and Salil P. Vadhan. Extracting randomness from samplable distributions. In *FOCS*, pages 32–42. IEEE Computer Society, 2000.

31 Salil P. Vadhan and Colin Jia Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19-22, 2012*, pages 817–836, 2012.

32 Andrew C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS'82, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society.

On the NP-Completeness of the Minimum Circuit Size Problem*

John M. Hitchcock[1] and A. Pavan[2]

1 **Department of Computer Science, University of Wyoming, US**
jhitchco@cs.uwyo.edu
2 **Department of Computer Science, Iowa State University, US**
pavan@cs.iastate.edu

Abstract

We study the Minimum Circuit Size Problem (MCSP): given the truth-table of a Boolean function f and a number k, does there exist a Boolean circuit of size at most k computing f? This is a fundamental NP problem that is not known to be NP-complete. Previous work has studied consequences of the NP-completeness of MCSP. We extend this work and consider whether MCSP may be complete for NP under more powerful reductions. We also show that NP-completeness of MCSP allows for amplification of circuit complexity. We show the following results.

- If MCSP is NP-complete via many-one reductions, the following circuit complexity amplification result holds: If NP $\cap$ co-NP requires $2^{n^{\Omega(1)}}$-size circuits, then E^{NP} requires $2^{\Omega(n)}$-size circuits.
- If MCSP is NP-complete under truth-table reductions, then EXP $\neq$ NP $\cap$ SIZE(2^{n^ϵ}) for some $\epsilon > 0$ and EXP $\neq$ ZPP. This result extends to polylog Turing reductions.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Minimum Circuit Size, NP-completeness, truth-table reductions, circuit complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.236

1 Introduction

Many natural NP problems are known to be NP-complete. Ladner's theorem [14] tells us that if P is different from NP, then there are NP-intermediate problems: problems that are in NP, not in P, but also not NP-complete. The examples arising out of Ladner's theorem come from diagonalization and are not natural. A canonical candidate example of a natural NP-intermediate problem is the Graph Isomorphism (GI) problem. If GI is NP-complete, then the polynomial-time hierarchy collapses [17, 8]. This gives very strong evidence that GI is unlikely to be NP-complete.

In this paper, we study another candidate example of NP-intermediate problem—the Minimum Circuit Size Problem (MCSP): given the truth-table of a Boolean function f and a number k, does there exist a Boolean circuit of size at most k computing f? We do not have a good understanding of the complexity of this fundamental problem. Clearly MCSP is in NP. It is believed that MCSP is not in P, however we do not know whether it is NP-complete. Unlike the GI problem, we do not currently have complexity-theoretic evidence that MCSP is not NP-complete. If MCSP is not NP-complete, then it implies that P does

* Research Supported in part by NSF grants 0917417, 0916797, and 1421163

not equal NP. Previous work has considered whether MCSP (and its variants) is complete via various notions of reductions [13, 16, 4]. These works establish that if MCSP is complete, then certain consequences happen for complexity classes – some plausible, some not. These results indicate that settling whether MCSP is NP-complete is outside the scope of current techniques.

Kabanets and Cai [13] showed that if MCSP is NP-complete under *natural* reductions, then (i) $E \not\subseteq P/poly$ and (ii) E requires $2^{\Omega(n)}$-size circuits or NP can be solved in subexponential time. On the contrary, they obtained a host of interesting consequences under the assumption that MCSP is in P. For example, they showed that if MCSP is in P, then Blum integers can be factored in time 2^{n^ϵ}. They also related this assumption to *circuit complexity amplification*. They showed that the assumption "MCSP is in P" yields the following: if there exists a language in E with circuit complexity $2^{\delta n}$ (for some $\delta > 0$), then there is a language in E with essentially maximal circuit complexity (close to $2^n/n$). Such a circuit complexity amplification result, even though believable, is surprising.

Recently Murray and Williams [16] showed that MCSP is not complete under *local reductions* where each output bit of the reduction can be computed in time $n^{1/2-\epsilon}$. They also showed that if MCSP is complete via AC_0 reductions then E has languages with circuit complexity $2^{\delta n}$. For the case of polynomial-time reductions, they showed that if MCSP is NP-complete via polynomial-time reductions,then $EXP \not\subseteq P/poly$ or EXP = NEXP. In particular, it follows that $EXP \neq NP \cap P/poly$ and $EXP \neq ZPP$. Even though we strongly believe that statements such as EXP differs from ZPP and E has high circuit complexity hold, we are far away from proving them. These results explain the difficult of proving a NP-completeness result for MCSP (if it is indeed NP-complete). Allender, Holden, and Kabanets studied the oracle version of MCSP problem. Given the truth-table of a Boolean function f and a parameter k, does f admit circuits of size k that have access to an oracle A? They showed that $MCSP^{QBF}$ is unlikely to be hard for various complexity classes under reductions that are more restrictive than polynomial-time reductions. For example, they showed that if $MCSP^{QBF}$ is hard for NP under logspace reductions, then nondeterministic exponential time (NEXP) collapses to $PSPACE$.

The known results that concern NP-completeness of MCSP and oracle versions of MCSP can be summarized as follows: For the case of "restricted" reductions, these problems are unlikely to complete NP-hard; Establishing NP-hardness under polynomial-time reductions would resolve a few major open problems in complexity theory.

Our Results. In this paper we obtain additional results regarding NP-completeness of MCSP under polynomial-time reductions. Our first result relates completeness of MCSP with *circuit complexity amplification*. If a complexity class $\mathcal{C}$ requires superpolynomial-size circuits, then can we amplify this hardness to show that a complexity class $\mathcal{D}$ requires circuits of much higher size? Ideally, we want $\mathcal{D}$ to be the same as $\mathcal{C}$. However, we do not know how to prove such results even when the class $\mathcal{D}$ is a superclass of $\mathcal{C}$. Buresh-Oppenheim and Santhanam [10] showed that if the *nondeterministic* circuit complexity of E is $2^{\delta n}$, then $E/O(n)$ has languages with maximal circuit complexity. They established a negative result that shows that known proof techniques can not amplify deterministic circuit complexity. As our first result, we show that NP-completeness of MCSP implies certain circuit complexity amplification. Assume that MCSP is NP-complete and suppose further that we have a moderately exponential-size ($2^{n^{\Omega(1)}}$) circuit-size lower bound for $NP \cap$ co-NP. We show that this hardness can be amplified into a strongly exponential ($2^{\Omega(n)}$) circuit-size lower bound for

E^{NP}. Admittedly, the gap between these classes is large, but we know of no unconditional method of doing this. This result should be contrasted with the previously mentioned result of Kabanets and Cai. Interestingly, both the statements "MCSP is in P" and "MCSP is NP-complete" imply that circuit complexity amplification is possible.

The statement "If NP ∩ co-NP requires circuits of size $2^{n^{\Omega(1)}}$, then E^{NP} requires circuits of size $2^{\Omega(n)}$" can also be viewed as an *upward separation result*—if a complexity class is hard, then a higher complexity class is much harder. In general, such upward separation results are rare. For example, we do not know if NP differs from P, then NEXP differs from EXP. Thus NP-completeness of MCSP implies an upward separation result.

Next we consider the completeness of MCSP under reductions that are more general than polynomial-time, many-one reductions. We do not know whether GI polynomial-time, many-one reduces to MCSP, however Allender and Das [3] showed that GI reduces to MCSP if we allow *probabilistic, Turing reductions*. These reductions use randomness and are allowed to ask multiple (adaptive) queries. This result suggests that allowing more general reductions to MCSP yields more power. This raises the following question: Is it possible to establish completeness of MCSP under more general reductions? In our second result, we show that it would be difficult to establish completeness MCSP under *truth-table/nonadaptive reductions*. We show that if MCSP is NP-complete under truth-table reductions, then EXP ≠ NP ∩ SIZE(2^{n^ϵ}) for some $\epsilon > 0$. This is an extension of Murray and Williams' result. We first provide an alternate proof of Murray and Williams result for the case of many-one reductions using different techniques, and extend this proof to the case of truth-table reductions. Our alternate proof could be of independent interest. We also note that the proof of Murray and Williams can also be extended to the case of truth-table reductions. Additionally, our proof extends to polylog-Turing reductions. It is worth noting that our results for truth-table completeness and polylog-Turing completeness are not directly comparable.

Techniques. Our approaches are based on ideas from honest reductions. A many-one reduction f is honest if $|f(x)| \geq |x|^\epsilon$ for some $\epsilon > 0$. From early work of Berman and Hartmanis [7], we know that all natural NP-complete problems are complete under honest reductions. Let f be a many-one reduction from L to MCSP. We say that this reduction is *parametric honest* if there is an $\epsilon > 0$ such that for every x, the output $f(x) = \langle y, k\rangle$ satisfies $k > n^\epsilon$. Note that L reduces to MCSP via honest reductions does not imply that L reduces to MCSP via parametric honest reductions. Suppose that MCSP is NP-complete via parametric honest reductions. Consider such a reduction f from 0^* to MCSP. Note that $f(1^n) = \langle y, k\rangle$ is a negative instance of MCSP, and since $k > n^\epsilon$, we have that the circuit complexity of y (when viewed a truth-table of a boolean function) is at least n^ϵ. Thus we have a polynomial-time algorithm that outputs strings with high circuit complexity which in turn implies that E has high circuit complexity.

We show that under certain plausible hypotheses, NP-completeness of MCSP implies that there exist parametric honest reductions to MCSP. We combine this with the above observation to obtain our results.

- In our first result, the hypothesis is that NP ∩ co-NP requires moderately exponential-size circuits. We show this implies MCSP is complete under parametric honest, SNP (strong nondeterministic) reductions. Informally, a reduction is an SNP reduction if it is computable by a NP ∩ co-NP machine. This yields the strong exponential-size circuit lower bound for E^{NP}.

- In our second result, the hypothesis is that there is a hard tally language T in NP. Using this hypothesis, we show that if MCSP is truth-table complete, then there is a truth-table reduction from T to MCSP where at least one query is parametric, honest. This yields a circuit lower bound for E. We build on this to show that truth-table completeness of MCSP implies a separation of EXP from ZPP.

Even though we know that all known NP-complete sets are complete via honest reductions, we do not whether this is true for all NP-complete sets. In recent years there have been a few results that show that, under some believable hypotheses, every NP-complete set is complete via honest reductions whose resource bounds are slightly larger than polynomial [2, 12, 11, 9]. We use ideas from these works to show that if MCSP is complete, then it is complete via parametric honest reductions (under certain hypotheses).

This paper is organized as follows. Section 2 covers preliminaries and previous work. Our results on truth-table completeness of MCSP are in Section 3. The consequences for amplification of circuit complexity are in section 4.

2 Preliminaries

For the standard notation and notions in complexity theory we refer the reader to [6]. Our alphabet is $\Sigma = \{0,1\}$ and we use Σ^n to denote all binary strings of length n. Given an n bit string (where n is a power of 2) x, we view x as the truth-table of a function, denoted f_x, from $\{0,1\}^{\log n}$ to $\{0,1\}$. Given a function $f : \Sigma^n \to \{0,1\}$, we use $CC(f)$ to denote the size of the smallest Boolean circuit that computes f. For a string x (whose length is a power of two), we use $CC(x)$ to denote $CC(f_x)$. For a language L, $L(x) = 1$ if $x \in L$; otherwise $L(x) = 0$. Given a language L, we use $L_n : \Sigma^n \to \{0,1\}$ to denote the characteristic function of L restricted to strings of length n. We say that $CC(L) > s(n)$ if there exist infinitely many n for which $CC(L_n) > s(n)$. We say that $CC(L) > s(n)$ a.e. if $CC(L_n) > s(n)$ for all but finitely many n. A complexity class $\mathcal{C}$ does not have circuits of size $s(n)$ if there exists $L \in \mathcal{C}$ such that $CC(L) > s(n)$.

▶ **Definition 2.1.** MCSP is the set of tuples $\langle x, k\rangle$ such that $CC(f_x)$ is at most k.

An instance $\langle x, k\rangle$ of MCSP is called ℓ-*large* is if $k \geq \ell$. In our proofs we use *strong nondeterministic reductions* [1, 15] and approximable sets. We define these notions.

▶ **Definition 2.2.** A language A reduces to a language B via strong, nondeterministic, polynomial-time reductions (SNP reductions), if there exists a polynomial-time bounded, nondeterministic Turing machine N such that for every $x \in \Sigma^*$, the following conditions hold:

- Every path of $N(x)$ outputs a string y or outputs a special symbol $\perp$.
- If $x \in A$, then every output y of $N(x)$ belongs to B; if $x \notin A$, every output y of $N(x)$ does not belong to B.

▶ **Definition 2.3.** A language L is $t(n)$-time 2-approximable [5], if there is exists a function f computable in time $O(t(n))$ such that for every pair of strings x and y, $f(x,y) \neq L(x)L(y)$. A language L is io-lengthwise, $t(n)$-time, 2-approximable if there exists a $O(t(n))$-time computable function f such that for infinitely many n for every pair of strings x and y of length n, $f(x,y) \neq L(x)L(y)$.

It is known that every polynomial-time, 2-approximable set has polynomial-size circuits [5]. This proof can be extended.

▶ **Theorem 2.4** ([5]). *If a language L is* io*-length wise, $t(n)$-time 2-approximable, then for infinitely many n, $CC(L_n) \leq O(t^2(n))$.*

▶ **Definition 2.5.** A language A is polynomial-time, truth-table reduces to a language B if there exist a pair of polynomial-time computable functions f and g such that for every x, $A(x) = f(x, B(q_1), \cdots, B(q_m))$, where $g(x) = \langle q_1, \cdots q_m \rangle$.

▶ **Definition 2.6.** Let L be a language that polynomial-time, many-one reduces to MCSP. We say that L reduces to MCSP via *parametric, honest reduction* if there exists an $\epsilon > 0$, and a polynomial-time, many-one reduction f from L to MCSP if $f(x)$ is $|x|^\epsilon$ large for every $x \in \Sigma^*$.

The above definition can be extended to the case of SNP reductions.

▶ **Definition 2.7.** We say that a language L reduces to MCSP via *parametric, honest,* SNP *reduction*, if there exists an $\epsilon > 0$ and a polynomial-time nondeterministic machine N such that L SNP reduces to MCSP via N and every output of $N(x)$, that does not equal $\perp$, is $|x|^\epsilon$-large.

The following observations are proved using the standard techniques.

▶ **Observation 2.8.** *Suppose that there is a* $\mathrm{P}/O(\log n)$ *algorithm A and an $\epsilon > 0$ such that for all but infinitely many n the output of $A(1^n)$ has circuit complexity greater than n^ϵ. Then there is a language L is* E *such that $CC(L) \geq 2^{\delta n}$ for some $\delta > 0$.*

▶ **Observation 2.9.** *Suppose that there is a non-deterministic, polynomial-time algorithm A and an $\epsilon > 0$ such that for infinitely many n the following holds: Every output of $A(1^n)$ that does not equal $\perp$, has circuit complexity greater than n^ϵ. Then there is a language L is* E^{NP} *such that $CC(L) \geq 2^{\delta n}$ for some $\delta > 0$.*

3 Amplification of Circuit Complexity

In this section we show that completeness of MCSP implies that circuit complexity can be amplified.

▶ **Theorem 3.1.** *Assume that* MCSP *is* NP*-complete via polynomial-time, many-one reductions. If there exists a language L in* $\mathrm{NP} \cap \text{co-NP}$ *such that for some $\epsilon > 0$, $CC(L) \geq 2^{n^\epsilon}$ a.e., then there exists $\delta > 0$ such that then* E^{NP} *does not have circuits of size $2^{\delta n}$.*

Before we proceed with proof, we give a brief overview of the proof. Gu, Hitchcock, and Pavan [11] showed that if NP can not be solved in sub-exponential time (at all lengths), then every NP-complete set is complete via P/poly, length-increasing reductions. We borrow ideas from this work. Let L be a hard language in $\mathrm{NP} \cap \text{co-NP}$ whose circuit complexity is high.

Our first step in the proof is that under this hypothesis, completeness of MCSP implies completeness via parametric, honest reductions. For this we define an intermediate language I that embeds both SAT and L. This language consists of tuples $\langle x, y, z \rangle$ so that $Maj(x \in L, y \in \mathrm{SAT}, z \in L)$ is 1. This language is clearly in NP. Consider a reduction f from I to MCSP. Suppose that $f(\langle x, y, z \rangle) = \langle u, k \rangle$. If k is small (less than n^δ), then we can solve the membership of $\langle u, k \rangle$ in time roughly 2^{n^δ}. If $\langle u, k \rangle$ is in MCSP then $\langle x, y, z \rangle \in I$. Thus it must be the case that at least one of x or z are in L. Thus $L(x)L(z)$ cannot be equal to 00. Thus in time 2^{n^δ} time we learned some information about the collective membership of x

and z in L (even though this information does help us solve individual memberships of x and z in L). Now suppose that for every pair x and z, we have that $f(\langle x, y, z\rangle)$ is small, then for every pair of strings x and z we can exclude one possibility for $L(x)L(z)$ in time 2^{n^δ}. This implies that L must be io-2-approximable and thus L has low enough circuit complexity (by Theorem 2.4). From this we conclude that for at least one pair x and z, $f(\langle x, y, z\rangle)$ is large. Using this we build a parametric, honest reduction from SAT to I. We now proceed with details.

Proof. Let L be a language in NP $\cap$ co-NP that does not have 2^{n^ϵ}-size circuits at almost all lengths. We will first prove that if MCSP is NP-complete, then MCSP is complete via parametric, honest, SNP reductions.

▶ **Lemma 3.2.** *Suppose that there exists a language in* NP $\cap$ co-NP *that requires* 2^{n^ϵ}*-size circuits a.e. for some* $\epsilon > 0$*. If* MCSP *is* NP*-complete, then* MCSP *is complete via parametric, honest,* SNP *reductions.*

Proof. Let L be the hard language in NP $\cap$ co-NP that requires 2^{n^ϵ}-size circuits. We define the following intermediate language I. Let $\delta = \epsilon/2$.

$$I = \{\langle x, y, z\rangle \mid Maj\{x \in L, y \in \text{SAT}, z \in L\} = 1, |x| = |z| = |y|^{1/\delta}\}.$$

Clearly I is in NP. Let f be a many-one reduction from I to MCSP. Our goal is to exhibit a large query, SNP reduction from SAT to MCSP. For this we will first show that for every string y of length n^δ, there exist $x \in L$, $z \notin L$ (of length n) such that $f(\langle x, y, z\rangle)$ is n^δ-large.

Let

$$T_n = \{\langle x, z\rangle \mid |x| = |z| = n, L(x) \neq L(z), \forall y \in \Sigma^{n^\delta}, f(\langle x, y, z\rangle) \text{ is not } n^\delta\text{-large}\}.$$

We will next claim that T_n must be the empty set for all but finitely many n.

▶ **Claim 3.2.1.** *For all but finitely many* n*,* $T_n = \emptyset$*.*

Proof. We prove by contradiction. Suppose that there exist infinitely many n at which T_n is not empty. We show that for infinitely many lengths n, $CC(L_n) \leq 2^{n^\epsilon}$, which contradicts the hardness of L. This contradiction is achieved by showing that L is io-lengthwise, 2-approximable in time 2^{n^ϵ}. Consider the following approximator function h:

1. Input: x, z of length n.
2. For every y from Σ^{n^δ} compute $f(\langle x, y, z\rangle)$.
3. If every $f(\langle x, y, z\rangle)$ is n^δ-large, then output 01 and stop.
4. If for some $y \in \Sigma^{n^\delta}$, $f(x, y, z)$ is not n^δ large, compute the membership of $f(x, y, z)$ in MCSP.
5. If $f(x, y, z) \in$ MCSP, then output 00; otherwise output 11.

Let n be a length at which $T_n \neq \emptyset$. We show that for every x, z of length n the output of the above algorithm does not equal $L(x)L(z)$. Since T_n is not empty, there exists a $y \in \{0,1\}^{n^\delta}$ such that $f(\langle x, y, z\rangle$ is not n^δ large. Thus the above algorithm reaches Step 4. If $f(x, y, z) \in$ MCSP, then the algorithm outputs 00. In this case, since f is a many-one reduction from I to MCSP, $\langle x, y, z\rangle \in I$. Thus at least one of x or z must belong to L. Thus $L(x)L(z) \neq 00$. Similarly, if $f(x, y, z) \notin$ MCSP, then $\langle x, y, z\rangle \notin I$, and this implies that at least one of x or z does not belong to L. Thus the output of the algorithm 11 does not equal $L(x)L(z)$.

We now bound the running time of the above algorithm. Step 2 takes $O(2^{n^\delta} \cdot poly(n))$ time. Consider Step 4. This step is performed only when $f(x, y, z) = \langle u, k\rangle$ is not n^δ-large. Thus $k \leq n^\delta$. Thus to decide the membership of $\langle u, k\rangle$, we have to cycle through all circuits of size $\leq n^\delta$ and check if any of them computes the function f_u. This step takes $2^{O(\log n n^\delta)}$ time. Thus the total time taken by the above algorithm is bounded by $2^{O(\log n n^\delta)}$.

If T_n is not empty for infinitely many n, the language L is io-lengthwise, 2-approximable in time $2^{O(\log n n^\delta)}$. Thus by Theorem 2.4, $CC(L_n) \leq 2^{n^\epsilon}$ for infinitely many n as $\delta \leq \epsilon/2$.This is a contradiction. ◀

We will now return to the proof of Lemma 3.2. Thus $T_n \neq \emptyset$ for all but finitely many lengths n. This suggests the following SNP reduction from SAT to MCSP: On an input y of length n, guess a string $x \in L$ and a string $z \notin L$ of lengths $n^{1/\delta}$ and compute $f(\langle x, y, z\rangle) = \langle u, k\rangle$. If $k < n$ output $\perp$, otherwise output $\langle u, k\rangle$. By claim 3.2.1, for all but finitely many n, $T_{n^{1/\delta}}$ is not empty. Thus for all but finitely many n, there exist strings x and z of length $n^{1/\delta}$ such that $x \in L$, $z \notin L$ and $f(\langle x, y, z\rangle)$ is n-large for every y of length n. Since L is in NP $\cap$ co-NP, at least one path of the reduction guesses such x and z and the output along this path is n-large. Thus MCSP is complete via parametric, honest, SNP-reductions. ◀

We now complete the proof of Theorem 3.1. Let $T = 0^*$, by Lemma 3.2, there is a SNP reduction f from T to MCSP that is parametric honest. Let $x_n = \langle y_n, k\rangle$ be the lexicographically smallest output produced by f on input 1^n. Since $1^n \notin T$, we have that $\langle y_n, k\rangle \notin$ MCSP and $k \geq n^\delta$. Thus $CC(y_n) \geq n^\delta$. By Observation 2.9, it follows that E^{NP} has high circuit complexity. ◀

4 Truth-Table Completeness

Our results in this section are based on the following hypothesis.

> *Hypothesis H:* There exists an $\epsilon > 0$ and a tally language in NP that cannot be solved deterministically in time 2^{n^ϵ}.

Before moving on to more powerful reductions, we begin by examining the case of many-one reducibility.

▶ **Theorem 4.1.** *Assume that Hypothesis H holds. If MCSP is NP-complete via polynomial-time, many-one reductions, then there exists a $\delta > 0$ such that* $\mathrm{E} \not\subseteq \mathrm{SIZE}(2^{\delta n})$.

Proof. Assume that MCSP is NP-complete and let T be the hard tally language that is not in DTIME(2^{n^ϵ}). Let f be a many-one reduction from T to MCSP. Fix $\delta < \epsilon$.

▶ **Claim 4.1.1.** *There exist infinitely many n such that $0^n \notin T$ and $f(0^n)$ is n^δ-large.*

Proof. Suppose not. For all but finitely many n at which $0^n \notin T$ we have that $f(0^n)$ is not δ-large. This means that if $f(0^n)$ is n^δ-large for some n, then $0^n \in T$. This suggests the following algorithm for T: On input 0^n, compute $f(0^n) = \langle x, k\rangle$. If $f(0^n)$ is n^δ-large, then accept 0^n. Otherwise, we have that $k < n^\delta$. Now cycle through all circuits of size at most k to determine the membership of $\langle x, k\rangle$ in MCSP. This lets us decide the membership of 0^n in T.

The time taken for this procedure is dominated by the time taken to cycle through all circuits of size at most k. Since there are at most $2^{O(\log n n^\delta)}$ such circuits, the language T can be decided in time less than 2^{n^ϵ}. This is a contradiction. ◀

Now consider the following polynomial-time algorithm that on input 0^n computes $f(0^n) = \langle x, k\rangle$ and outputs x. Note that for infinitely many n, this algorithm outputs the truth-table of a function whose circuit complexity is at least n^δ. This implies that there is a language in E whose circuit complexity is $2^{\delta n}$. ◀

The above theorem yields the following corollary, similar to Murray and Williams [16]. The consequence here EXP $\neq$ NP $\cap$ SIZE(2^{n^ϵ}) is stronger than EXP $\neq$ NP $\cap$ P/poly obtained by Murray and Williams, though we note that their proof may be adapted to obtain this as well.

▶ **Corollary 4.2.** *If* MCSP *is* NP*-complete, then* EXP $\neq$ ZPP *and* EXP $\neq$ NP $\cap$ SIZE(2^{n^ϵ}) *for some* $\epsilon > 0$.

Proof. Assume that MCSP is NP-complete. We consider two cases.

- If Hypothesis H does not hold, then NP $\neq$ EXP as EXP has tally languages that can not be solved in time 2^n. Since ZPP is a subset of NP, EXP $\neq$ ZPP.
- If Hypothesis H holds, then by the above theorem, E does not have circuits of size $2^{\delta n}$ (at infinitely many lengths). This implies that ZPP can be derandomized to P at infinitely many length and which in turn implies that EXP $\neq$ ZPP. Finally note that, if E does not have circuits of size $2^{\delta n}$, then EXP does not have circuits of size 2^{n^ϵ} for some $\epsilon > 0$.

In both cases, the conclusion of the corollary is true. ◀

Next we extend the above theorem (and its proof) to the case of truth-table reductions. We note that the proof of Murray and Williams can also be extended to the case of truth-table reductions.

▶ **Theorem 4.3.** *Assume that the hypothesis H holds. If* MCSP *is truth-table complete for* NP*,* E *does not have circuits of size* $2^{\delta n}$ *for some* $\delta > 0$.

Proof. Let T be the hard tally language in NP and let f a truth-table reduction from T to MCSP. On input 0^n, let $q_1^n, \cdots q_m^n$ be the queries produced by f. Fix $\delta < \epsilon$. We first claim that at least one of the queries produced is large and is a negative instance of MCSP.

▶ **Claim 4.3.1.** *There exist infinitely many* n *for which there exists* i, $1 \leq i \leq m$, *such that* q_i^n *is* n^δ*-large and does not belong to* MCSP*.*

Proof. Suppose not. For all but finitely many n, the following holds. For every i, $1 \leq i \leq m$, either q_i^n is not n^δ-large or $q_i^n \in$ MCSP. This suggests the following algorithm to decide T:

On input 0^n, run the reduction f and produce queries $q_1^n, \cdots q_m^n$.
- If q_i^n is not n^δ-large then use a brute-force search algorithm to decide the membership of q_i^n in MCSP.
- If q_i^n is n^δ-large, then $q_i^n \in$ MCSP.

Use all answers to the queries decide the membership of 0^n in T.

Clearly, the algorithm correctly decides T. The most expensive step of the algorithm is to decide the membership of q_i^n in MCSP using the brute-force algorithm. Note that we run the brute-force algorithm only when q_i^n is not n^δ-large. Thus the time taken for this step is $2^{O(\log n n^\delta)}$. Thus the total time taken by the algorithm is $O(m2^{O(\log n n^\delta)})$. Since m is polynomial in n and $\delta < \epsilon$, this is bounded by 2^{n^ϵ}. This contradicts our hypothesis. ◀

Using the above claim, we show that there is an efficient algorithm (with a logarithmic amount of advice) that outputs strings with high circuit complexity.

▶ **Claim 4.3.2.** *There is a* $\mathrm{P}/O(\log n)$ *algorithm* $\mathcal{A}$ *that on input* 0^n *outputs a string* x_n *and for infinitely many* n, $CC(x_n) \geq n^\delta$.

Proof. Let n^ℓ bound the run time of the truth-table reduction from T to MCSP. The algorithm on input 0^n gets a tuple $\langle b, r\rangle$ as advice where b is a bit and $r < n^\ell$. The bit b is set to 1 if at least one of q_i^n is n^δ-large and does not belong to MCSP; otherwise b is set to 0. When b is 1, then the number r indicates the first index i, $1 \leq i \leq m$, for which q_i^n is n^δ-large and does not belong to MCSP. When b equals 0, r is set to 0. Note that the length of the advice is $O(\log n)$.

The algorithm on input 0^n first looks at the advice bit b. If b is 0, then it outputs 0^n. Otherwise it runs the reduction from T to MCSP to produce queries $q_1^n, \cdots q_m^n$. Let $q_r^n = \langle x_n, k\rangle$. The algorithm outputs x_n.

By Claim 4.3.1, there exist infinitely many n at which at least one of q_i^n is n^δ-large and does not belong to MCSP. At every such length the above algorithm (on correct advice bits) outputs a string x_n for which $CC(x_n) > n^\delta$. ◀

By Observation 2.8, there is a language in E that requires circuits of size $2^{\rho n}$ for some $\rho > 0$. This completes the proof of the theorem. ◀

As before we have the following corollary.

▶ **Corollary 4.4.** *If* MCSP *is truth-table complete for* NP*, then* EXP $\neq$ ZPP *and* EXP $\neq$ NP $\cap$ SIZE(2^{n^ϵ}) *for some* $\epsilon > 0$.

Using similar ideas we can prove the following.

▶ **Theorem 4.5.** *If* MCSP *is polylog-Turing complete for* NP*, then* EXP $\neq$ ZPP *and* EXP $\neq$ NP $\cap$ SIZE(2^{n^ϵ}) *for some* $\epsilon > 0$.

References

1 L. Adleman and K. Manders. Reducibility, randomness, and intractability. In *Proc. 9th ACM Symp. Theory of Computing*, pages 151–163, 1977.

2 M. Agrawal. Pseudo-random generators and structure of complete degrees. In *17th Annual IEEE Conference on Computational Complexity*, pages 139–145, 2002.

3 E. Allender and B. Das. Zero knowledge and circuit minimization. In *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 25–32, 2014.

4 E. Allender, D. Holden, and V. Kabanets. The minimum oracle circuit size problem. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 21–33, 2015.

5 A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and non-uniform complexity. *Inf. Comput.*, 186(1):104–139, 2003.

6 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

7 L. Berman and H. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.*, 6:305–322, 1977.

8 R. Boppanna, J. Hastad, and S. Zachos. Does Co-NP have short interactive proofs? *Information Processing Letters*, 25(2):125–132, 1987.

9 H. Buhrman, B. Hescott, S. Homer, and L. Torenvliet. Non-uniform reductions. *Theory Comput. Syst.*, 47(2):317–341, 2010.

10 J. Bursesh-Oppenheim and R. Santhanam. Making hard problems harder. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 73–87, 2006.

11 X. Gu, J. M. Hitchcock, and A. Pavan. Collapsing and separating completeness notions under average-case and worst-case hypotheses. *Theory of Computing Systems*, 51(2):248–265, 2011.

12 J. M. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. *Information and Computation*, 205(5):694–706, 2007.

13 V. Kabanets and J. Y. Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 73–79, 2000.

14 R. Ladner. On the structure of polynomial time reducibility. *J. Assoc. Comput. Mach.*, 22:155–171, 1975.

15 T. Long. Strong nondeterministic polynomial-time reducibilities. *Theor. Comput. Sci.*, 21:1–25, 1982.

16 C. Murray and R. Williams. On the (non) np-hardness of computing circuit complexity. In *Computational Complexity Conference*, 2015.

17 U. Schöning. The power of counting. In A. Selman, editor, *Complexity Theory Retrospective*, pages 204–223. Springer-Verlag, 1990.

Counting Euler Tours in Undirected Bounded Treewidth Graphs

Nikhil Balaji[1,3], Samir Datta[1], and Venkatesh Ganesan[2]

1 **Chennai Mathematical Institute, Chennai, India**
{nikhil,sdatta}@cmi.ac.in
2 **Birla Institute of Technology and Science – Pilani, India**
venkatesh920@gmail.com
3 **Indian Institute of Technology, Bombay, India**
nbalaji@cse.iitb.ac.in

Abstract

We show that counting Euler tours in undirected bounded tree-width graphs is tractable even in parallel - by proving a $\#\mathsf{SAC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{P}$ upper bound. This is in stark contrast to #P-completeness of the same problem in general graphs.

Our main technical contribution is to show how (an instance of) dynamic programming on bounded *clique-width* graphs can be performed efficiently in parallel. Thus we show that the sequential result of Espelage, Gurski and Wanke [16] for efficiently computing Hamiltonian paths in bounded clique-width graphs can be adapted in the parallel setting to count the number of Hamiltonian paths which in turn is a tool for counting the number of Euler tours in bounded tree-width graphs. Our technique also yields parallel algorithms for counting longest paths and bipartite perfect matchings in bounded-clique width graphs.

While establishing that counting Euler tours in bounded tree-width graphs can be computed by non-uniform monotone arithmetic circuits of polynomial degree (which characterize $\#\mathsf{SAC}^1$) is relatively easy, establishing a uniform $\#\mathsf{SAC}^1$ bound needs a careful use of polynomial interpolation.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, G.2.2 Graph Theory

Keywords and phrases Euler Tours, Bounded Treewidth, Bounded clique-width, Hamiltonian cycles, Parallel algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.246

1 Introduction

An Euler tour of a graph is a closed walk on the graph that traverses every edge in the graph exactly once. Given a graph, deciding if there is an Euler tour of the graph is quite simple. Indeed, the famous Königsberg bridge problem that founded graph theory is a question about the existence of an Euler tour using each of these bridges exactly once. Euler settled this question in the negative and in the process gave a necessary and sufficient condition for a graph to be *Eulerian* (A connected graph is Eulerian if and only if all the vertices are of even degree). This gives a simple algorithm to check if a graph is Eulerian.

An equally natural question is to ask for the number of distinct Euler tours in a graph. For the case of directed graphs, the BEST theorem due to De Bruijn, Ehrenfest, Smith and Tutte gives an exact formula that gives the number of Euler tours in a directed graph [1, 25] which yields a polynomial time algorithm via a determinant computation. For undirected graphs,

no such closed form expression is known and the computational problem is #P-complete [8]. In fact, the problem is #P-complete even when restricted to 4-regular planar graphs [18]. So exactly computing the number of Euler tours is not in polynomial time unless $\#\mathsf{P} = \mathsf{P}$.

In this paper, we are concerned with the problem of counting Euler tours on graphs of bounded treewidth. Many problems which are NP-hard for general graphs, can be solved in polynomial time on bounded treewidth graphs. Indeed, a result of Courcelle [11] asserts that any graph property that is expressible in Monadic Second Order logic (with edge quantifiers) can be solved in linear time on bounded treewidth graphs. Elberfeld et al. [15] adapt the theorem of Courcelle in the parallel setting and prove a L bound. However, Eulerianity is provably not MSO -expressible [14] and hence the approaches mentioned above are not directly applicable in our context.

Our strategy to count Euler tours is as follows: Given a bounded treewidth graph G, we count the number of Euler tours of G by counting the number of Hamiltonian tours of the line graph of G, $L(G)$. In general, there is no bijection between these two quantities, but we show that G can be modified to obtain G' ($tw(G') \leq tw(G) + 3$) such that G, G' have the same number of Eulerian tours, which equals the number of Hamiltonian tours of $L(G')$. Henceforth, we will be primarily interested in line graphs of bounded treewidth graphs. It is known that such graphs are of bounded clique-width [24]

We base our proof on a proof that the decision version of Hamiltonicity is polynomial time computable in bounded clique-width graphs [16]. We prove that this algorithm can be parallelised and extended to the counting version. Next, we show that for line graphs of bounded tree-width graphs which form the family of interest, the clique-width expression can be inferred from the corresponding tree decomposition. The tree decomposition itself is obtainable by the L-version of Bodlaender's theorem [15].

Our main tool in establishing a uniform NC-bound for counting Hamiltonian cycles on bounded clique-width graphs hinges on *polynomial interpolation*. While polynomial interpolation has been used successfully to compute various graph polynomials [23], our use is somewhat indirect and subtle: it is used by the uniformity machine to populate a table whose entries do not depend on the input bounded clique-width expression but only the number of vertices in the corresponding graph and the clique-width. We then build a monotone arithmetic circuit that uses the clique-width expression of the graph and entries from this table to count the number of Hamiltonian cycles in the clique-width bounded graph. We then observe that since the number of distinct Hamiltonian tours of a graph is at most exponential in the number of vertices of the graph, and the circuit is monotone, the formal degree of the circuit must be a polynomial in the size of the input graph. This allows us to use a result from circuit complexity [2] to yield an upper bound of $\#\mathsf{SAC}^1$ on the complexity of counting Euler tours on bounded treewidth graphs.

Our techniques also yield a parallel upper bound on the problems of counting longest paths/cycles and counting bipartite matchings in bounded clique-width graphs. These are well known problems (and #P-complete in general graphs) but their (counting) complexity has not been investigated in bounded clique-width graphs. While [13] studies the problem of counting longest paths and perfect matchings in bounded *tree-width DAGs*, we improve the results by resolving the problems for bounded *clique-width graphs* at the cost of replacing the L bound by a $\#\mathsf{SAC}^1$ bound where we know that $\mathsf{L} \subseteq \#\mathsf{SAC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{P}$ [1].

[1] Note that $\#\mathsf{SAC}^1$ is a function class and when we say $\#\mathsf{SAC}^1 \subseteq \mathsf{NC}^2$, what we actually mean is that any bit of the $\#\mathsf{SAC}^1$ function family of interest is computable by a NC^2 circuit family

1.1 Previous Work

Chebolu, Cryan, Martin have given a polynomial time algorithm for counting Euler tours in undirected series-parallel graphs [9] and they have claimed to extend it to a polynomial time algorithm [10] for the counting Euler tours in bounded tree-width graphs. We would like to point out that the only incomplete, unrefereed manuscript available publicly [10] sketches an algorithm that does dynamic programming directly on the tree-decomposition. Since we show how to obtain the line graph of the bounded tree-width graph efficiently in parallel and then work on this bounded clique-width graph - our approach is fundamentally different from that of [9, 10]. Another difference is that their algorithm is not designed to be parallelisable.

Also notice that in a precursor to this paper [4], using totally different techniques (basically applications of the Logspace version of Courcelle's theorem [15]) it was claimed that the number of Euler tours in bounded tree-width directed and undirected graphs can be counted in Logspace but the approach had a serious flaw in the undirected version. Later versions [5, 6] of the paper claim the result only for directed graphs. This work proves a slightly weaker version of the result - the upper bound being $\#\mathsf{SAC}^1$ rather than Logspace.

Given that counting Hamiltonian cycles on bounded clique-width graphs will suffice for our purposes, one result that is directly relevant is that of Flarup and Lyaudet [17]: They study the expressive power of Perfect Matching and Hamiltonian polynomials of graphs of bounded clique-width and show that they can simulate arithmetic polynomials, and are themselves contained in VP. This yields a GapSAC^1 bound (implicit) for counting Hamiltonian cycles in bounded clique-width graphs right away. There are two aspects in which the work of [17] differs from our work: Firstly, even though their techniques are also inspired from [16] like ours, they work with a slightly different notion of clique-width namely $W-m-$clique-width[2]. Secondly, in the case of counting Euler tours, from a straight-forward application of [17] the best upper bound that can be obtained from the circuit families constructed in [17] is non-uniform GapSAC^1, whereas we get an upper bound of Logspace-uniform[3] $\#\mathsf{SAC}^1$.

There is some similarity that this work bears with that of Makowsky et al. [23], in that both involve polynomial interpolation to count witnesses for a graph theory problem. The similarity is somewhat superficial because we use interpolation to obtain numbers *independent* of the input graph while they interpolate to compute a graph polynomial that crucially depends on the graph. The choice of graph theory problems is also quite different. In particular, [23] does not address the Hamiltonian cycle problem.

1.2 Our Results

This is the main theorem of this work:

▶ **Theorem 1.** *#Hamiltonian Cycles (or Paths) for bounded clique-width graphs is in* $\#\mathsf{SAC}^1$. *Consequently, #Euler Tours for bounded tree-width graphs is also in* $\#\mathsf{SAC}^1$.

As a bonus we also get the following:

▶ **Theorem 2.** *The following counts can be obtained in* $\#\mathsf{SAC}^1$ *for bounded clique-width graphs (given a bounded clique-width expression for the graph):*

---

[2] These are weighted versions of clique-width and are used to produce weighted graphs. [17] motivate this variant of clique-width by observing that since K_n has clique-width 2, most graph polynomials are VNP-complete for bounded clique-width graphs.

[3] In an earlier version of this paper, we had erroneously claimed a GapL upper bound for counting Euler tours. As pointed out to us by Ramprasad Saptharishi, there is a rather serious gap with this approach.

1. *#Hamiltonian Cycles*
2. *#Longest Paths/Cycles*
3. *#Cycle Covers*
4. *#Perfect Matchings (for bipartite graphs)*

1.3 Overview of Algorithm

Every Euler tour in a graph yields a Hamiltonian cycle in its line graph. Though this map is not bijective we show that we can make it so by altering the input graph slightly. It is well known [20] that the line graphs of bounded tree-width graphs have bounded clique-width. We show how to obtain a bounded clique-width decomposition for the line graph of a bounded tree-width graph in Logspace using the Logspace version of Courcelle's Theorem [15] by first obtaining a bounded tree-width decomposition via a Logspace version of Bodlaender's theorem [15].

Our main algorithm replaces the sequential procedure from [16] to decide if a bounded clique-width graph has a Hamiltonian path. Instead, it computes the number of Hamiltonian cycles. The procedure uses elementary counting coupled with polynomial interpolation to compute some matrices which are independent of the input graph depending only on its size. The matrices are then combined with vectors maintaining counts, along the structure tree of the clique-decomposition. A degree bound for the monotone arithmetic circuit then suffices to prove the $\#\mathsf{SAC}^1$ bound.

1.4 Organization

The rest of the paper is organized as follows: In Section 2, we introduce some definitions and results that will be helpful in understanding the rest of the paper. Section 3 shows how to obtain a clique-width expression for the line graph of a bounded treewidth graph in Logspace. Section 4 presents a $\#\mathsf{SAC}^1$ implementation of our algorithm to count the number of Hamiltonian tours in graphs of bounded clique-width. We conclude with some unresolved questions related to this work in Section 5.

2 Preliminaries

▶ **Definition 3** (Line Graph). For an undirected graph $G = (V, E)$, the line graph of G denoted $L(G) = (L_V, L_E)$ is the graph where $L_V = E$ and $(e_i, e_j) \in L_E$ if and only if there exists a vertex $v \in V$ such that both e_i and e_j are incident on v.

▶ **Definition 4** (Treewidth). Given an undirected graph $G = (V_G, E_G)$ a tree decomposition of G is a tree $T = (V_T, E_T)$(the vertices in $V_T \subseteq 2^{V_G}$ are called *bags*), such that

1. Every vertex $v \in V_G$ is present in at least one bag, i.e., $\cup_{X \in V_T} X = V_G$.
2. If $v \in V_G$ is present in bags $X_i, X_j \in V_T$, then v is present in every bag X_k in the unique path between X_i and X_j in the tree T.
3. For every edge $(u, v) \in E_G$, there is a bag $X_r \in V_T$ such that $u, v \in X_r$.

The width of a tree decomposition is $\max_{X \in V_T}(|X| - 1)$. The treewidth of a graph is the minimum width over all possible tree decomposition of the graph.

▶ **Definition 5** (NLC-width). Let k be a positive integer. The class NLC_k of labeled graphs $G = (V, E, lab_G)$ where $lab_G : V \to [k]$, is recursively defined as follows:

1. The single vertex graph labeled by a label a, $\bullet_a$ for $a \in [k]$ is in NLC_k.

2. Let $G = (V_G, E_G, lab_G) \in NLC_k$ and $H = (V_H, E_H, lab_H) \in NLC_k$ be two vertex-disjoint labeled graphs and $S \subseteq [k]^2$, then $G \times_S H = (V', E', lab') \in NLC_k$, where $V' = V_G \cup V_H$ and

$$E' = E_G \cup E_H \cup \{(u, v) | u \in V_G, v \in V_H, (lab_G(u), lab_H(v)) \in S\}$$

and for all $u \in V'$,

$$lab'(u) = \begin{cases} lab_G(u), & \text{if } u \in V_G \\ lab_H(u), & \text{if } u \in V_H \end{cases}$$

3. Let $G = (V_G, E_G, lab) \in NLC_k$ and $R : [k] \to [k]$ be a function, then $\circ_R(G) := (V_G, E_G, lab')$ defined by $lab'(u) = R(lab(u))$ for all $u \in V_G$ is in NLC_k.

The NLC-width[4] of a labeled graph G is the least integer k such that $G \in NLC_k$. An expression Y built with $\bullet_a, \times_S, \circ_R$, for integers $a \in [k]$, $S \in [k]^2$ and $R : [k] \to [k]$ is called a NLC-width k expression. The graph defined by expression Y is denoted by $val(Y)$.

▶ **Definition 6** (Clique Width)**.** Let k be a positive integer. The class CW_k of labeled graphs $G = (V, E, lab_G)$ where $lab_G : V \to [k]$ is recursively defined as follows:

1. The single vertex graph labeled by a label a, $\bullet_a$ for $a \in [k]$ is in CW_k.
2. Let $G = (V_G, E_G, lab_G) \in CW_k$ and $H = (V_H, E_H, lab_H) \in CW_k$ be two vertex-disjoint labeled graphs. Then $G \oplus H = (V', E', lab') \in CW_k$, where $V' = V_G \cup V_H$ and $E' = E_G \cup E_H$ and for all $u \in V'$

$$lab'(u) = \begin{cases} lab_G(u), & \text{if } u \in V_G \\ lab_H(u), & \text{if } u \in V_H \end{cases}$$

3. Let a, b be distinct positive integers and $G = (V_G, E_G, lab) \in CW_k$ be a labeled graph. Then,
   (a) $\rho_{a \to b}(G) := (V_G, E_G, lab') \in CW_k$ where for all $u \in V_G$

$$lab'(u) = \begin{cases} lab_G(u), & \text{if } lab_G(u) \neq a \\ b, & \text{if } lab_G(u) = a \end{cases}$$

   (b) $\eta_{a,b}(G) := (V_G, E', lab_G) \in CW_k$ where,

$$E' = E_G \cup \{(u, v) | u, v \in V_G, lab(u) = a, lab(v) = b\}$$

The clique-width of a labeled graph G is the least integer k such that $G \in CW_k$. An expression X built with $\bullet_a, \oplus, \rho_{a \to b}, \eta_{a,b}$ for integers $a, b \in [k]$ is called a clique-width k expression. By $val(X)$, we denote the graph defined by expression X.

▶ **Definition 7** (Chordal graph, Chordal completion)**.** A graph is said to be chordal if every cycle with at least 4 vertices always contains a chord. A chordal completion of a graph G is a chordal graph with the same vertex set as G which contains all edges of G.

▶ **Definition 8** (Perfect Elimination Ordering, Elimination Tree [19])**.** Let $G = (V, E)$ be a graph and $o = (v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of G. Let $N^-(G, o, i)$ and

[4] NLC stands for *Node Label Controlled*, has its origins in graph grammars, was defined by Wanke [28].

$N^+(G, o, i)$ for $i = 1, \ldots, n$ be the set of neighbors v_j of vertex v_i with $j < i$ and $j > i$ respectively.

$$\begin{aligned} N^-(G, o, i) &= \{v_j | (v_i, v_j) \in E \text{ and } j < i\} \\ N^+(G, o, i) &= \{v_j | (v_i, v_j) \in E \text{ and } j > i\} \end{aligned}$$

The vertex order o is said to be a Perfect Elimination Ordering (PEO) if for all $i \in [n]$, $N^+(G, o, i)$ induces a complete subgraph of G. The structure of G can then be characterized by a tree $T(G, o) = (V_T, E_T)$ defined as follows:

$$\begin{aligned} V_T &= V \\ E_T &= \{(v_i, v_j) \in E | i < j \text{ and } \forall j', i < j' < j, (v_i, v_{j'}) \notin E\} \end{aligned}$$

Such a $T(G, o)$ is called the Elimination Tree associated with the graph G.

For more information on Chordal graphs and PEO, we refer the reader to Golumbic's book [19].

▶ **Definition 9** (Cycle Cover)**.** A *cycle cover* $\mathcal{C}$ of $G = (V, E)$ is a set of vertex-disjoint cycles that cover the vertices of G. I.e., $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where $V(C_i) = \{c_{i_1}, \ldots, c_{i_{r(i)}}\} \subseteq V$ such that (c_{i_1}, c_{i_2}), (c_{i_2}, c_{i_3}), …, $(c_{i_{r(i)-1}}, c_{i_{r(i)}})$, $(c_{i_{r(i)}}, c_{i_1}) \in E(C_i) \subseteq E$ and $\uplus_{i=1}^{k} V(C_i) = V$. The least numbered vertex $h_i \in V(C_i)$, is called the head of the cycle.

▶ **Definition 10** (#SAC^1)**.** #SAC^1 is the class of functions from $\{0,1\}^n$ to nonnegative integers computed by polynomial-size logarithmic-depth, semi unbounded arithmetic circuits[5], using $+$ (unbounded fan-in) and $\times$ gates (fan-in 2) and the constants 0 and 1.

For further background on circuit complexity, we refer the reader to [27].

▶ **Proposition 11** ([2, 26])**.** *Any function $f : \{0,1\}^n \to R$, where R is a semi-ring, computed by arithmetic circuits of size s and degree d can be computed by semi-unbounded arithmetic circuits of size* poly(s, d) *and depth $O(\log d)$. In particular, all functions computed by polynomial sized circuits of polynomial degree are exactly those in* #SAC^1*.*

▶ **Fact 12** (Kronecker substitution [12])**.** *Let $P(x_1, x_2, \ldots, x_n)$ be a multivariate polynomial of degree d. We replace every occurence of variable x_i by x^{d^i}. This yields an unique univariate polynomial $Q(x)$ of degree at most $d^{O(n)}$ such that P can be efficiently recovered from the knowledge of coefficients of Q. When the number of variables is a constant, the degree of the multivariate polynomial and the univariate polynomial are polynomially related.*

3 From Euler Tours to Hamiltonian cycles

It is possible to construct a graph G such that G has no Eulerian tours, but $L(G)$ has a Hamiltonian cycle[6]. Proposition 13 gives necessary and sufficient conditions for when a line graph of a given graph is Hamiltonian.

▶ **Proposition 13** ([21])**.** *$L(G)$ is Hamiltonian if and only if G has a closed trail that contains at least one end point of every edge.*

[5] Note that such circuits have degree that is at most a polynomial in the number of input variables.
[6] Indeed, there is a 2-connected graph – K_4 with one of the edges removed – which is non-Eulerian but its line graph is Hamiltonian.

Given a graph G, we want to construct a graph G' such that every closed trail in G' that contains at least one end point of every edge is exactly an Eulerian tour of G'. The following Lemma guarantees exactly this:

▶ **Lemma 14.** *Given an undirected graph G, construct a graph $G' = (V', E')$ from G as follows: Replace every edge $e = (u, v)$ of G by path of length three. Then G and G' have the same number of Eulerian tours and the Eulerian tours of G' are in bijection with the Hamiltonian tours of $L(G')$.*

Notice that G is a minor of G', and the tree decomposition of G' can be obtained from that of G by locally adding to each bag containing an edge e of G, the extra vertices and edges of the path of length three. Hence, the following is immediate:

▶ **Proposition 15.** *G has bounded treewidth iff G' has bounded treewidth.*

▶ **Proposition 16** ([20])**.** *If G is of treewidth k, then $L(G)$ has clique-width $f(k) = 2k + 2$.*

▶ **Proposition 17** ([15])**.** *Given a bounded treewidth graph G, a balanced tree decomposition*[7] *of G is obtainable in* L*.*

We first need the Perfect Elimination Ordering(PEO) of the vertices of the graph. It is known that a graph has a PEO if and only if it is chordal. Since we can do a chordal completion of a bounded treewidth graph (while preserving treewidth), such an ordering of the vertices always exists. Recently Arvind et al. gave a Logspace procedure for obtaining a PEO in k-trees (which are maximal treewidth-k graphs). We adapt this for graphs that are chordal completions of bounded treewidth graphs:

▶ **Lemma 18** (Adapted from [3])**.** *Given a balanced tree decomposition of a bounded treewidth graph G, a Perfect Elimination Ordering and the corresponding elimination tree of a chordal completion of G, which is a balanced binary tree of depth $O(\log n)$, can be computed in* L*.*

▶ **Lemma 19** (Adapted from [20])**.** *Given the tree decomposition of a graph G along with a elimination tree, the clique-width expression X of $L(G)$ is obtainable in* L*. The parse tree of this clique-width expression has height at most $O(\log n)$*

We show in the subsequent Lemma that the method in [20] is amenable to a Logspace implementation when provided with a PEO of the vertices of the graph.

▶ **Lemma 20** (Adapted from [20])**.** *The NLC-width of the line graph $L(G)$ of a graph G of treewidth k is at most $k + 2$ and such a NLC-width expression is obtainable in* L*.*

Gurski and Wanke [20] observe that it is sufficient to look at G that are k-trees here because the line graph of every subgraph of G then is an induced subgraph of the line graph of G and the class NLC_k is closed under taking induced subgraphs for every $k \geq 1$ (See Theorem 4 in [20]). Our method involves dealing with bounded treewidth graphs that are chordal, which are a strict superclass of k-trees and we observe that the property mentioned above still holds in this case.

▶ **Proposition 21.** *Given a graph G of NLC-width at most k by an NLC-width expression Y, we can obtain the clique-width expression X of G, where $|X| \leq 2k + 2$ in* L*.*

[7] A tree decomposition of a graph is said to be balanced if the tree underlying the decomposition is balanced

To sum up, these are the main preprocessing steps:

1. Obtain a balanced binary tree decomposition of the input treewidth k graph G in Logspace via Proposition 17 [15].
2. Obtain the tree decomposition of G' (as required by Proposition 13 and specified by Lemma 14) from the tree decomposition of G.
3. Perform a chordal completion of G' by adding edges to every bag.
4. Obtain a PEO tree of G' of height $O(\log n)$, where every vertex has at most k children via Lemma 18.
5. Construct a NLC width $(k+2)$ expression for $L(G')$ via Lemma 20
6. From the NLC width $(k+2)$ expression, construct a clique-width $(2k+2)$ expression for $L(G')$ via Proposition 21 (The surplus edges added during the chordal completion are removed at this step).

The proofs of Lemma 14, 18, 19, 20 and Proposition 21 can be found in the full version of the paper [7].

4 The #SAC^1 upper bound

Let X be the clique-width k expression for a labeled graph $G = (V, E, lab)$ such that G is val(X) and let $|V| = n$. Let G be of clique-width k. Hence by Definition 6, G can be constructed from the graph with n isolated labeled vertices, using at most k labels. Notice that X can be viewed as a tree (we will refer to this as the parse tree of the clique-width expression) with the n isolated labeled vertices at the leaves and every internal node is labeled with one of the operations $o = \{\bullet_i, \oplus, \eta_{i,j}, \rho_{i\to j} : i, j \in [k] \wedge i \neq j\}$ To each internal vertex of the tree, we can associate a graph (possibly disconnected) which is a subgraph of G, and at the root of the tree, we get G itself. The size of the tree is polynomial in n and k. Our objective in this section will be to count the number of Hamiltonian cycles in G, when provided with the clique-width expression X. We will count along the parse tree of the clique-width expression.

To this end, we call a subset of edges $E' \subseteq E$ *path-cycle covers*, if in the subgraph $G' = (V, E', lab)$ every vertex in G' has degree at most 2. To every such G', we associate a multiset M consisting of multisets $\langle lab(v_1), lab(v_r)\rangle$ one each for every path/cycle $p = v_1, \ldots, v_r$, $r \geq 1$, in G', where v_1, v_r have degree at most 1 in G' if they exist (p being a cycle otherwise). Let $F(X)$ be the set of all multisets M for all such subsets $E' \subseteq E$.

Let K be the set of all possible labels of the end points, in the labeled graph produced at the output of each node in the parse tree. We refer to elements of K as *types*. Note that every M consists of at most $|K|$ distinct *types* and $F(X)$ has at most $(n+1)^{|K|}$ distinct multisets each with at most n multisets of size 2. Here $K = K_0 \uplus K_1 \uplus K_2$ is the set of distinct *types* where K_2 accounts for types of the form $\langle i, j\rangle$ (for $i \neq j$) corresponds to paths whose end points are i and j; K_0 for the empty type $\langle\rangle = \emptyset$ corresponds to a *cycle*; K_1 for types of the form $\langle i, i\rangle$ which could be either paths whose end points are both labeled i, or isolated vertices with the label i. Observe that, $|K_2| = \binom{k}{2}$, $K_1 = 2k$ and $K_0 = 1$, where we distinguish between the cases of single isolated vertex of label i and multiple vertex paths with end points labeled i for technical reasons, leading to the extra factor of 2. Our notation is consistent with [16] in all cases except for the empty type, since in [16] cycles are not permitted.

Our objective is to count the number of path-cycle covers, $\#X[M]$, corresponding to a multiset M in the graph $val(X)$. In particular,

$$\sum_{i,j\in[K]} \#X[M_{i,j}]$$

where $M_{i,j} = \langle\langle i,j\rangle\rangle$ is a multiset containing a single type $\langle i,j\rangle$, yields the number of Hamiltonian paths with end points coloured i,j in $val(X)$. We denote by $\#X$ the vector indexed by M and hence has $(n+1)^K$ entries where $\#X[M]$ (where $M \in [0,n]^K$) stores the count of the number of path/cycle covers of type specified by M in the graph $val(X)$. Let C_o be a $(n+1)^K \times (n+1)^K$ matrix which for each pair of multisets M, M' denotes the number of ways to form M' from M under an operation $o \in \{\eta_{i,j}, \rho_{i\to j} : i,j \in [k] \wedge i \neq j\}$. C_o is defined uniquely for the two kinds of operations η, ρ and is independent of the input graph val(X).

Then the following is an easy consequence of the definitions:

▶ **Proposition 22.** *The value of $\#X$ is given by:*
1. *if $X = \bullet_i$ then if $M = \langle\langle i,i\rangle\rangle$ then $\#X[M] = 1$; else $\#X[M] = 0$.*
2. *else if $X = X_1 \oplus X_2$ then*

$$\#X[M] = \sum_{M'\in[0,n]^K : M'\subseteq M} \#X_1[M']\#X_2[M \setminus M']$$

3. *else if $X = \rho_{i\to j}(X_1)$ then $(C_{\rho_{i\to j}})^T \#X_1$*
4. *else $X = \eta_{i,j}(X_1)$ then $(C_{\eta_{i,j}})^T \#X_1$*

Proof. The first item is immediate. For the second, notice that each multiset of types M in the disjoint union of two graphs is formed by picking multisets M', M'' from the two graphs respectively and taking their multi-union. Thus the number of distinct ways to form M is obtained by considering all possible decompositions of M into sets M', M'' one from each graph. Since, this is a decomposition $M'' = M \setminus M'$, the correctness of the second item follows.

For the third and the fourth items, notice that we have a matrix C such that $C[M, M']$ is the number of ways to convert a multiset M to a multiset M'. Thus the number of ways to form M' is to take the product of $\#X[M]C[M, M']$ and add up the products over all M. This is the stated form in matrix notation. ◀

Proposition 22 enables us to prove the $\#\mathsf{SAC}^1$ upper bound:

▶ **Lemma 23.** *For a bounded clique-width expression X, for every multiset of types, M, the value $\#X[M]$ of the number of path-cycle covers at any node along the parse tree of the clique-width expression can be computed in $\#\mathsf{SAC}^0$ where the inputs to the $\#\mathsf{SAC}^0$ circuit are entries of the matrix C_o for $o \in \{\bullet_i, \oplus, \eta_{i,j}, \rho_{i\to j} : i,j \in [k] \wedge i \neq j\}$. The number of path-cycle covers in the input graph can hence be counted in $\#\mathsf{SAC}^1$.*

The proof of Lemma 23 can be found in the full version of the paper [7]. We now turn to the proof of our main Theorem 1

Proof. (of Theorem 1) To count Euler tours on bounded treewidth graphs, we can count Hamiltonian cycles in the line graph (via Lemma 14). Here we need to compute the quantity $\#X[\langle\emptyset\rangle]$ (since the empty multiset represents a cycle, the path-cycle cover consisting of a single cycle must be a Hamiltonian cycle itself). This follows from Lemma 23. ◀

Proof. (of Theorem 2) Hamiltonian cycles can be counted in $\#\mathsf{SAC}^1$ by Lemma 23. Longest Cycles (Paths) can be counted by considering multisets which consist of a single cycle (respectively, path) and the minimum number of isolated vertices respectively. To see this observe that for every cycle (respectively, path) C in the graph there is a multiset consisting of a single empty type (respectively, non-empty type) and $|V(G)| - |V(C)|$ isolated vertices respectively.

Counting cycle covers is equally simple. We just need to add up the counts for multisets consisting only of empty types. This, is of course because an empty type represents a cycle.

Perfect Matchings in bipartite graphs can therefore be counted by counting the cycle covers in a biadjacency matrix. ◀

4.1 Computing $C_{\rho_{i\to j}}$ and $C_{\eta_{i,j}}$

It is easy to compute $C_{\rho_{i\to j}}$ by the following,

▶ **Proposition 24.** *$C(\rho_{i\to j})$ is a $\{0,1\}$-matrix such that the entry corresponding to M_1, M_2 is equal to 1 iff $\rho_{i\to j}(M_1) = M_2$ (it is 0 otherwise).*

Let $W_{\vec{\alpha}}(t')$ denote the number of ways to form one path/cycle of type $t' \in K$, given a multiset of paths/cycles consisting of $\vec{\alpha}(t)$ paths/cycles for every type $t \in K$.

Next, we show how to compute $C_{\eta_{i,j}}$:

▶ **Lemma 25.** *There is a Logspace Turing machine that takes input $W_{\vec{\alpha}}(t')$ for every $\vec{\alpha} \in [0,n]^{|K|}, t' \in K$ and outputs the entries of the matrix $C_{\eta_{i,j}}$.*

Proof. We show that each entry can be computed in DLOGTIME-uniform TC^0 which is contained in L (see e.g. Vollmer [27]). Our main tool in this lemma is an application of polynomial interpolation.

Notice that the rows/columns of the matrix C are indexed by multisets of *types*. Here a *type* is an element from K. Therefore any such multiset can be described by a vector $\vec{\alpha}$ of length $|K|$. Here each entry of the vector represents the number of paths/cycles with that type inside the multiset.

In the following we will consistently make use of the notation, $\vec{z}^{\vec{a}}$ to denote: $\prod_{i\in I} z_i^{a_i}$, where I is the index set for both $\vec{z}, \vec{a}$.

We have the following:

▶ **Lemma 26.** *$C[M, M']$ is the coefficient of $\vec{x}^{\vec{c'}}\vec{y}^{\vec{c}}$ in the following polynomial $p_{\vec{c'},\vec{c}}(\vec{x},\vec{y})$:*

$$\prod_{t,t'\in K}\prod_{\vec{\alpha}\in[0,n]^{|K|}}\sum_{\vec{d}_{\vec{\alpha}}}\left(W_{\vec{\alpha}}(t')x_{t'}y_{t'}^{\alpha(t)}\right)^{d_{\vec{\alpha}}(t')}$$

To fix the notation we reiterate (items 1, 2, 3, 4 were defined previously and we introduce some new notation in items 5, 6, 7, 8, 9):

1. K is the set of types, where $|K| = \binom{k}{2} + 2k + 1$.
2. $t, t' \in K$ are types in the input, output multiset (respectively M, M').
3. A *allocation*, $\alpha(t) \in [0,n]$ is the number of path-cycle covers of type $t \in K$.
4. $\vec{\alpha} \in [0,n]^{|K|}$ is a possible allocation vector indexed by K in which each entry is $\alpha(t)$.
5. $d_{\vec{\alpha}}(t') \in [0,n]$ is the number of paths of type t' formed from each allocation of type $\vec{\alpha}$.
6. $\vec{d}_{\vec{\alpha}} \in [0,n]^{|K|}$ is a vector indexed by K in which each entry is one of $d_{\vec{\alpha}}(t')$.
7. $W_{\vec{\alpha}}(t')$ is the number of ways to form a single path/cycle of type t' from an allocation vector $\vec{\alpha}$.

8. $\vec{W}_{\vec{\alpha}}$ is the vector indexed by K in which each entry is one of $W_{\vec{\alpha}}(t')$.
9. $\vec{c}, \vec{c'} \in [0,n]^{|K|}$ are vectors indicating number of paths/cycles in M, M' respectively.
To see that Lemma 25 follows from Lemma 26 we use Kronecker substitution (see Fact 12) to convert the multivariate polynomial $p_{\vec{c'},\vec{c}}(\vec{x},\vec{y})$ with $2|K|$ variables to a univariate polynomial. Then we use Lagrange interpolation to find the coefficient of an arbitrary term - in particular, the term corresponding to $\vec{x}^{\vec{c'}}\vec{y}^{\vec{c}}$ in TC^0 (see e.g. Corollary 6.5 in [22]). ◀

Proof. (of Lemma 26) Consider the following expression:

$$\sum_{\vec{d}\in\mathbf{D}} \prod_{t'\in K} \prod_{\vec{\alpha}\in[0,n]^{|K|}} W_{\vec{\alpha}}(t')^{d_{\vec{\alpha}}(t')} \tag{1}$$

where the sum is taken over $\mathbf{D} \subseteq [0,n]^{|K|}$ consisting of all $\vec{d}$'s satisfying:

$$\forall t', \sum_{\vec{\alpha}\in[0,n]^{|K|}} d_{\vec{\alpha}(t')} = c'(t') \tag{2}$$

$$\forall t, \sum_{\vec{\alpha}\in[0,n]^{|K|}} \sum_{t'\in K} \alpha(t) d_{\vec{\alpha}}(t') = c(t) \tag{3}$$

▶ **Claim 27.** *$C[M,M']$ equals Expression (1).*

Proof. (of Claim) The Condition 2 above asserts that the number of paths/cycles of type t' present in M' equals the sum over all $\vec{\alpha}$ of the number of paths/cycles of type t' using resources described by $\vec{\alpha}$; the Condition 3 is essentially a conservation of resource equation for every type t saying that all the resources present in M are used one way or the other in M'.

Let P, P' be path-cycle covers represented by M, M' respectively such that we can obtain P' from P, i.e., P' is one possible path-cycle cover that can be obtained by an $\eta_{i,j}$ operation on P. This transformation is described by a unique $\vec{d}$. Then the pair contributes precisely one to $C[M,M']$. On the other hand $\{P,P'\}$ satisfies (2),(3) so contributes exactly one to the summand corresponding to the unique $\vec{d}$ in Expression 1. Since the pair P, P' corresponds to a unique $\vec{d}$ and contributes exactly one, the remaining summands would evaluate to zero. This can be explained by observing that for all $\vec{d'} \neq \vec{d}$, the number of paths of type t in $\vec{d'}$ is not equal to the corresponding number in $\vec{d}$ for atleast one t. Hence, they would contribute nothing to pair P, P'. ◀

To complete the proof notice that the coefficient of $\vec{x}^{\vec{c'}}\vec{y}^{\vec{c}}$ is precisely expression 1 under the conditions 2, 3. Now, we explain the reasoning behind expression 1. We have $d_{\vec{\alpha}}(t')$ paths of type t', each of which can be formed in $W_{\vec{\alpha}}(t')$ ways. Note that each of these $d_{\vec{\alpha}}(t')$ paths are formed from different $\vec{\alpha}$ (though the values of each of these $d_{\vec{\alpha}}(t')$ vectors $\vec{\alpha}$ is the same, they are inherently different as they are composed of mutually exclusive vertex sets) and we consider each valid set of $d_{\vec{\alpha}}(t')$ vectors $\vec{\alpha}$, exactly once. Hence, we multiply with $W_{\vec{\alpha}}(t')^{d_{\vec{\alpha}}(t')}$ to get the final count. ◀

4.2 Calculation of $W_{\vec{\alpha}}(t')$

$W_{\vec{\alpha}}(t')$ denotes the number of ways to form exactly one type $t' \in K$ in M' given a multiset of types consisting of $\vec{\alpha}(t)$ types for every type $t \in K$ in M. For simplicity of notation, let $t = \langle i,j\rangle \in K$ be a type and let $\beta(i) = \alpha(\langle i,i\rangle) + \alpha^{(=0)}(\langle i,i\rangle)$ be the total number of multisets of type $\langle i,i\rangle$, where $\alpha(\langle i,i\rangle)$ (respectively $\alpha^{(=0)}(\langle i,i\rangle)$) denote paths (respectively single nodes) labeled $\langle i,i\rangle$ in $\vec{\alpha}$. Note that this distinction is not necessary for types where the end points have different labels.

▶ **Lemma 28.** *For an operation η_{i_0,j_0} in the clique-width expression and for any type $t' = \langle i, j\rangle$, $W_{\vec{\alpha}}(t')$ is given by*

$$W_{\vec{\alpha}}(\langle i,j\rangle) = [\![\langle i,j\rangle]\!]_{\vec{\alpha}} W_{\vec{\alpha}}$$

where,

$$W_{\vec{\alpha}} = \binom{\alpha(\langle i_0,j_0\rangle)+\beta(i_0)+\beta(j_0)}{\alpha(\langle i_0,j_0\rangle)} \alpha(\langle i_0,j_0\rangle)!\beta(i_0)!\beta(j_0)!2^{\alpha(\langle i_0,i_0\rangle)+\alpha(\langle j_0,j_0\rangle)}$$

and, $[\![\langle i,j\rangle]\!]_{\vec{\alpha}}$ is given by [8], [9]

- $[\![\langle a,b\rangle]\!]_{\vec{\alpha}} = [\![\beta(a) = \beta(b)]\!]$
- $[\![\langle a,a\rangle^{(=0)}]\!]_{\vec{\alpha}} = [\![\alpha_{a,a}^{(=0)} = 1 \wedge \alpha(\langle a,b\rangle) = 0]\!]$
- $[\![\langle a,a\rangle]\!]_{\vec{\alpha}} = [\![\beta(a) = \beta(b)+1]\!]$
- $[\![\langle i,a\rangle]\!]_{\vec{\alpha}} = [\![(\alpha(\langle i,a\rangle) = 1 \wedge \beta(a) = \beta(b)) \vee (\alpha(\langle i,b\rangle) = 1 \wedge \beta(a) = \beta(b)+1)]\!]$
- $[\![\langle i,j\rangle]\!]_{\vec{\alpha}} = [\![(\alpha(\langle i,a\rangle) = 1 \wedge \alpha(\langle a,j\rangle) = 1 \wedge \beta(a) = \beta(b)+1) \vee (\alpha(\langle i,a\rangle) = 1 \wedge \alpha(\langle b,j\rangle) = 1 \wedge \beta(a) = \beta(b))]\!]$
- $[\![\langle i,i\rangle]\!]_{\vec{\alpha}} = [\![(\alpha(\langle i,a\rangle) = 2 \wedge \beta(a) = \beta(b)+1) \vee (\alpha(\langle i,a\rangle) = 1 \wedge \alpha(\langle i,b\rangle) = 1 \wedge \beta(a) = \beta(b))]\!]$
- $[\![\langle\emptyset\rangle]\!]_{\vec{\alpha}} = [\![\beta(a) = \beta(b)]\!]$

where, $\{a,b\} = \{i_0,j_0\}$ in some order.

Proof. (of Lemma 28) Let's look at $W_{\vec{\alpha}}(\langle a,a\rangle)$ in detail. The $W_{\vec{\alpha}}(t)$ for all the other types t are computed similarly. Type $\langle a,a\rangle$ can be formed from the alternating sequence of types $\langle a,a\rangle, \langle b,b\rangle, \langle a,a\rangle \ldots \langle a,a\rangle$ interleaved with some (possibly zero) $\langle a,b\rangle$ types. Thus, the equality $\beta(a) = \beta(b)+1$ should hold while $\alpha(\langle a,b\rangle)$ can be any arbitrary non-negative integer. When $\alpha(\langle a,b\rangle) = 0$, the condition $\alpha(\langle a,a\rangle) \geq 1 \vee \alpha(\langle a,a\rangle)^{(=0)} > 1$ should hold to ensure that we are not considering the type $(\langle a,a\rangle)^{(=0)}$.

The number of ways of interspersing $\alpha(\langle a,b\rangle)$ types among $\beta(a)+\beta(b)$ types is

$$\binom{\alpha(\langle a,b\rangle)+\beta(a)+\beta(b)}{\alpha(\langle a,b\rangle)}$$

We can do this for all permutations of the $\langle a,b\rangle, \langle a,a\rangle$ and $\langle b,b\rangle$ types hence we multiply by: $\alpha(\langle a,b\rangle)!\beta(a)!\beta(b)!$. Finally, we can flip the orientation of paths of types $\langle a,a\rangle$ and $\langle b,b\rangle$ as they are equivalent respectively to their flipped orientations. Note that single nodes cannot be flipped. The proof is therefore completed by multiplying with: $2^{\alpha(\langle a,a\rangle)+\alpha(\langle b,b\rangle)}$. Lastly, a boundary case occurs when $\alpha(\langle a,b\rangle) = 0$ where every path can be flipped. Here, it is easy to see that in considering every permutation of types while accounting for flips, we end up counting each path twice (including its reverse). Hence, in this case we divide by 2. ◀

5 Conclusion and Open Ends

- Can the #SAC^1 bound be improved, to say, GapL or Logspace?
- How far can the Euler tour result be extended? To bounded clique-width graphs? Chordal graphs?
- Can the determinant of bounded clique-width adjacency matrices be computed in better than #SAC^1? (it is known to be L-hard even for bounded tree-width graphs from [6]).

[8] In this section, the notation $[\![S]\!]$ represents the Boolean value of the statement S. $[\![t]\!]_{\vec{\alpha}}$ represents a Boolean valued normalizing factor associated with the type t under the allocation vector $\vec{\alpha}$.

[9] We adopt a convention in which types t' (other than the type $\langle i_0,j_0\rangle$) not explicitly included in the expressions have an allocation $\alpha_{t'}$ equalling zero.

Acknowledgements. We would like to thank K. Narayan Kumar, Aniket Mane, M. Praveen and Prakash Saivasan for illuminating discussions regarding this paper. We would like to thank Eric Allender, Vikraman Arvind, Nutan Limaye, Meena Mahajan, Pierre McKenzie, Partha Mukhopadhyay, Ramprasad Saptharishi, Srikanth Srinivasan and V.Vinay for reading a follow-up work that resulted from this paper and for their comments, from which we discovered an error in a previous version of this paper.Thanks are also due to anonymous referees for several comments that helped improve the content and presentation of the paper. This work is partially funded by a grant from the Infosys Foundation.

References

1 van T. Aardenne-Ehrenfest and N. G. de Bruijn. Circuits and trees in oriented linear graphs. *Simon Stevin: Wis-en Natuurkundig Tijdschrift*, 28:203, 1951.

2 Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998.

3 Vikraman Arvind, Bireswar Das, Johannes Köbler, and Sebastian Kuhnert. The isomorphism problem for k-trees is complete for logspace. *Information and Computation*, 217:1–11, 2012.

4 Nikhil Balaji and Samir Datta. Tree-width and logspace: Determinants and counting Euler tours. *CoRR*, abs/1312.7468, 2013.

5 Nikhil Balaji and Samir Datta. Bounded treewidth and space-efficient linear algebra. *CoRR*, abs/1412.2470, 2014.

6 Nikhil Balaji and Samir Datta. Bounded treewidth and space-efficient linear algebra. In *Theory and Applications of Models of Computation*, pages 297–308. Springer, 2015.

7 Nikhil Balaji, Samir Datta, and Venkatesh Ganesan. Counting Euler tours in undirected bounded treewidth graphs. *arXiv:1510.04035v1 [cs.CC]*, 2015. `http://arxiv.org/abs/1510.04035v1`.

8 Graham Brightwell and Peter Winkler. Counting Eulerian circuits is #p-complete. In *ALENEX/ANALCO*, pages 259–262, 2005.

9 Prasad Chebolu, Mary Cryan, and Russell Martin. Exact counting of Euler tours for generalized series-parallel graphs. *J. Discrete Algorithms*, 10:110–122, 2012.

10 Prasad Chebolu, Mary Cryan, and Russell Martin. Exact counting of Euler tours for graphs of bounded treewidth. *CoRR*, abs/1310.0185, 2013.

11 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

12 David Cox, John Little, and Donal O'shea. *Ideals, varieties, and algorithms*, volume 3. Springer, 1992.

13 B. Das, S. Datta, and P. Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. *Theory Comput. Syst.*, 53(4):669–689, 2013.

14 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

15 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *FOCS*, pages 143–152, 2010.

16 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *WG*, pages 117–128. Springer, 2001.

17 Uffe Flarup and Laurent Lyaudet. On the expressive power of permanents and perfect matchings of matrices of bounded pathwidth/cliquewidth. *Theory of Computing Systems*, 46(4):761–791, 2010.

18 Qi Ge and Daniel Štefankovič. The complexity of counting Eulerian tours in 4-regular graphs. *Algorithmica*, 63(3):588–601, 2012.

19 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

20 Frank Gurski and Egon Wanke. Line graphs of bounded clique-width. *Discrete Mathematics*, 307(22):2734–2754, 2007.

21 Frank Harary and C St JA Nash-Williams. On Eulerian and Hamiltonian graphs and line graphs. *Canadian Mathematical Bulletin*, 8:701–709, 1965.

22 W. Hesse, E. Allender, and D.A.M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

23 J. A. Makowsky, U. Rotics, I. Averbouch, and B. Godlin. Computing graph polynomials on graphs of bounded clique-width. In *WG*, pages 191–204, 2006.

24 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

25 WT Tutte and CAB Smith. On unicursal paths in a network of degree 4. *The American Mathematical Monthly*, 48(4):233–237, 1941.

26 V. Vinay. Counting auxiliary pushdown automata. In *Structure in Complexity Theory*, pages 270–284, 1991.

27 Heribert Vollmer. *Introduction to circuit complexity – a uniform approach.* Texts in theoretical computer science. Springer, 1999.

28 Egon Wanke. k-nlc graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2):251–266, 1994.

Revisiting Robustness in Priced Timed Games

Shibashis Guha[1], Shankara Narayanan Krishna[2], Lakshmi Manasa[2], and Ashutosh Trivedi[2]

1 **Department of Computer Science & Engineering, IIT Delhi, India**
shibashis@cse.iitd.ernet.in
2 **Department of Computer Science & Engineering, IIT Bombay, India**
{krishnas,manasa,trivedi}@cse.iitb.ac.in

Abstract

Priced timed games are optimal-cost reachability games played between two players – the controller and the environment – by moving a token along the edges of infinite graphs of configurations of priced timed automata. The goal of the controller is to reach a given set of target locations as cheaply as possible, while the goal of the environment is the opposite. Priced timed games are known to be undecidable for timed automata with 3 or more clocks, while they are known to be decidable for automata with 1 clock. In an attempt to recover decidability for priced timed games Bouyer, Markey, and Sankur studied robust priced timed games where the environment has the power to slightly perturb delays proposed by the controller. Unfortunately, however, they showed that the natural problem of deciding the existence of optimal limit-strategy – optimal strategy of the controller where the perturbations tend to vanish in the limit – is undecidable with 10 or more clocks. In this paper we revisit this problem and improve our understanding of the decidability of these games. We show that the limit-strategy problem is already undecidable for a subclass of robust priced timed games with 5 or more clocks. On a positive side, we show the decidability of the existence of almost optimal strategies for the same subclass of one-clock robust priced timed games by adapting a classical construction by Bouyer at al. for one-clock priced timed games.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Priced Timed Games, Decidability, Optimal strategies, Robustness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.261

1 Introduction

Two-player zero-sum games on priced timed automata provide a mathematically elegant modeling framework for the control-program synthesis problem in real-time systems. In these games, two players – the *controller* and the *environment* – move a token along the edges of the infinite graph of configurations of a timed automaton to construct an infinite execution of the automaton in order to optimize a given performance criterion. The optimal strategy of the controller in such game then corresponds to control-program with the optimal performance. By priced timed games (PTGs) we refer to such games on priced timed automata with optimal reachability-cost objective. The problem of deciding the existence of the optimal controller strategy in PTGs is undecidable [8] with 3 or more clocks, while it is known to be decidable [5] for automata with 1 clock. Also, the ε-optimal strategies can be computed for priced timed games under the non-Zeno assumption [1, 4]. Unfortunately, however, the optimal controller strategies obtained as a result of solving games on timed automata may not be physically realizable due to unrealistic assumptions made in the modeling using timed automata, regarding the capability of the controller in enforcing precise delays. This severely

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 261–277

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

limits the application of priced timed games in control-program synthesis for real-time systems.

In order to overcome this limitation, Bouyer, Markey, and Sankur [7] argued the need for considering the existence of robust optimal strategies and introduced two different robustness semantics – *excess* and *conservative* – in priced timed games. The key assumption in their modeling is that the controller may not be able to apply an action at the exact time delays suggested by the optimal strategy. This phenomenon is modeled as a *perturbation* game where the time delay suggested by the controller can be perturbed by a bounded quantity. Notice that such a perturbation may result in the guard of the corresponding action being disabled. In the conservative semantics, it is the controller's responsibility to make sure that the guards are satisfied after the perturbation. On the other hand, in the excess semantics, the controller is supposed to make sure that the guard is satisfied before the perturbation: an action can be executed even when its guard is disabled ("excess") post perturbation and the valuations post perturbation will be reflected in the next state. The game based characterization for robustness in timed automata under "excess" semantics was first proposed by Bouyer, Markey, and Sankur [6] where they study the parameterized robust (qualitative) reachability problem and show it to be EXPTIME-complete. The "conservative" semantics were studied for reachability and Büchi objectives in [14] and shown to be PSPACE-complete. For a detailed survey on robustness in timed setting we refer to an excellent survey by Markey [12].

Bouyer, Markey, and Sankur [7] showed that the problem for deciding the existence of the optimal strategy is undecidable for priced timed games with 10 or more clocks under the excess semantics. In this paper we further improve the understanding of the decidability of these games. However, to keep the presentation simple, we restrict our attention to turn-based games under excess semantics. To further generalize the setting, we permit both positive and negative price rates with the restriction that the accumulated cost in any cycle is non-negative (akin to the standard no-negative-cycle restriction in shortest path game problems on finite graphs). We improve the undecidability result of [7] by proving that optimal reachability remains undecidable for robust priced timed automata with 5 clocks. Our second key result is that, for a fixed δ, the cost optimal reachability problem for one clock priced timed games with no-negative-cycle restriction is decidable for robust priced timed games with given bound on perturbations. To the best of our knowledge, this is the first decidability result known for robust timed games under the excess semantics. A closely related result is [9], where decidability is shown for robust timed games under the conservative semantics for a fixed δ.

2 Preliminaries

We write $\mathbb{R}$ for the set of reals and $\mathbb{Z}$ for the set of integers. Let $\mathcal{C}$ be a finite set of real-valued variables called *clocks*. A *valuation* on $\mathcal{C}$ is a function $\nu : \mathcal{C} \to \mathbb{R}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the clock with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{R}^{|\mathcal{C}|}$. Abusing notations slightly, we use a valuation on $\mathcal{C}$ and a point in $\mathbb{R}^{|\mathcal{C}|}$ interchangeably. For a subset of clocks $X \subseteq \mathcal{C}$ and valuation $\nu \in \mathbb{R}^{|\mathcal{C}|}$, we write $\nu[X{:=}0]$ for the valuation where $\nu[X{:=}0](x) = 0$ if $x \in X$, and $\nu[X{:=}0](x) = \nu(x)$ otherwise. The valuation $\mathbf{0} \in \mathbb{R}^{|\mathcal{C}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{C}$. A clock constraint over $\mathcal{C}$ is a subset of $\mathbb{R}^{|\mathcal{C}|}$. We say that a constraint is *rectangular* if it is a conjunction of a finite set of constraints of the form $x \bowtie k$, where $k \in \mathbb{Z}$, $x \in \mathcal{C}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. For a constraint $g \in \varphi(\mathcal{C})$, we write $[\![g]\!]$ for the set of valuations in $\mathbb{R}^{|\mathcal{C}|}$ satisfying g. We write $\varphi(\mathcal{C})$ for the set of rectangular constraints over $\mathcal{C}$.

We use the terms constraints and guards interchangeably.

Following [5] we introduce priced timed games with external cost function on target locations (see [10]). For this purpose, we define a *cost function*[5] as a piecewise affine continuous function $f : \mathbb{R}^n_{\geq 0} \to \mathbb{R} \cup \{+\infty, -\infty\}$. We write $\mathcal{F}$ for the set of all cost functions.

▶ **Definition 2.1** (Priced Timed Games)**.** A turn-based two player *priced timed game* is a tuple $\mathcal{G} = (L_1, L_2, L_{init}, \mathcal{C}, X, \eta, T, f_{goal})$ where L_i is a finite set of *locations* of Player i, $L_{init} \subseteq L_1 \cup L_2$(let $L_1 \cup L_2 = L$) is a set of initial locations, $\mathcal{C}$ is an (ordered) set of *clocks*, $X \subseteq L \times \varphi(\mathcal{C}) \times 2^{\mathcal{C}} \times (L \cup T)$ is the *transition relation*, $\eta\colon L \to \mathbb{Z}$ is the price function, T is the set of target locations, $T \cap L = \emptyset$; and $f_{goal} : T \to \mathcal{F}$ assigns external cost functions to target locations.

We refer to Player 1 as the controller and Player 2 as the environment. A priced timed game begins with a token placed on some initial location ℓ with valuation $\mathbf{0}$ and cost accumulated being so far being 0. At each round, the player who controls the current location ℓ chooses a delay t (to be elapsed in l) and an outgoing transition $e = (\ell, g, r, \ell') \in X$ to be taken after t delay at ℓ. The clock valuation is then updated according to the delay t, the reset r, the cost is incremented by $\eta(\ell) \cdot t$ and the token is moved to the location ℓ'. The two players continue moving the token in this fashion, and give rise to a sequence of locations and transitions called a *play* of the game. A configuration or state of a PTG is a tuple (ℓ, ν, c) where $\ell \in L$ is a location, $\nu \in \mathbb{R}^{|\mathcal{C}|}$ is a valuation, and c is the cost accumulated from the start of the play. We assume, w.l.o.g [2], that the clock valuations are bounded.

▶ **Definition 2.2** (PTG semantics)**.** The semantics of a PTG $\mathcal{G}$ is a labelled state-transition game arena $[\![\mathcal{G}]\!] = (\mathcal{S} = S_1 \uplus S_2, S_{init}, A, E, \pi, \kappa)$ where

- $S_j = L_j \times \mathbb{R}^{|\mathcal{C}|}$ are the Player j states with $\mathcal{S} = S_1 \uplus S_2$,
- $S_{init} \subseteq \mathcal{S}$ are initial states s.t. $(\ell, \nu) \in S_{init}$ if $\ell \in L_{init}$, $\nu = \mathbf{0}$,
- $A = \mathbb{R}_{\geq 0} \times X$ is the set of *timed moves*,
- $E : (\mathcal{S} \times A) \to \mathcal{S}$ is the transition function s.t. for $s = (\ell, \nu), s' = (\ell', \nu') \in \mathcal{S}$ and $\tau = (t, e) \in A$ the function $E(s, \tau)$ is defined if $e = (\ell, g, r, \ell')$ is a transition of the PTG and $\nu \in [\![g]\!]$; moreover $E(s, \tau) = s'$ if $\nu' = (\nu + t)[r{:=}0]$ (we write $s \xrightarrow{\tau} s'$ when $E(s, \tau) = s'$);
- $\pi : \mathcal{S} \times A \to \mathbb{R}$ is the price function such that $\pi((\ell, \nu), (t, e)) = \eta(\ell) \cdot t$; and
- $\kappa : \mathcal{S} \to \mathbb{R}$ is an external cost function such that $\kappa(\ell, \nu)$ is defined when $\ell \in T$ such that $\kappa(\ell, \nu) = f_{goal}(\ell)(\nu)$.

A *play* $\rho = \langle s_0, \tau_1, s_1, \tau_2, \ldots, s_n \rangle$ is a finite sequence of states and actions s.t. $s_0 \in S_{init}$ and $s_i \xrightarrow{\tau_{i+1}} s_{i+1}$ for all $0 \leq i < n$. The infinite plays are defined in an analogous manner. For a finite play ρ we write its last state as $\text{last}(\rho) = s_n$. For a (infinite or finite) play ρ we write $\text{stop}(\rho)$ for the index of first target state and if it doesn't visit a target state then $\text{stop}(\rho) = \infty$. We denote the set of plays as $\mathsf{Plays}_{\mathcal{G}}$. For a play $\rho = \langle s_0, (t_1, a_1), s_1, (t_2, a_2), \ldots \rangle$ if $\text{stop}(\rho) = n < \infty$ then $\mathsf{Cost}_{\mathcal{G}}(\rho) = \kappa(s_n) + \sum_{j=1}^{n} \pi(s_{i-1}, (t_i, a_i))$ else $\mathsf{Cost}_{\mathcal{G}}(\rho) = +\infty$.

A *strategy* of player j in $\mathcal{G}$ is a function $\sigma : \mathsf{Plays}_{\mathcal{G}} \to A$ such that for a play ρ the function $\sigma(\rho)$ is defined if $\text{last}(\rho) \in S_j$. We say that a strategy σ is memoryless if $\sigma(\rho) = \sigma(\rho')$ when $\text{last}(\rho) = \text{last}(\rho')$, otherwise we call it memoryful. We write Strat_1 and Strat_2 for the set of strategies of player 1 and 2, respectively.

A play ρ is said to be *compatible to a strategy* σ of player $j \in \{1, 2\}$ if for every state s_i in ρ that belongs to Player j, $s_{i+1} = \sigma(s_i)$. Given a pair of strategies $(\sigma_1, \sigma_2) \in \mathsf{Strat}_1 \times \mathsf{Strat}_2$, and a state s, the outcome of (σ_1, σ_2) from s denoted $\mathsf{Outcome}(s, \sigma_1, \sigma_2)$ is the unique play that starts at s and is compatible with both strategies. Given a player 1 strategy $\sigma_1 \in \mathsf{Strat}_1$

we define its cost $\mathsf{Cost}_{\mathcal{G}}(s, \sigma_1)$ as $\sup_{\sigma_2 \in \mathsf{Strat}_2}(\mathsf{Cost}(\mathsf{Outcome}(s, \sigma_1, \sigma_2)))$. We now define the *optimal reachability-cost* for Player 1 from a state s as

$$\mathsf{OptCost}_{\mathcal{G}}(s) = \inf_{\sigma_1 \in \mathsf{Strat}_1} \sup_{\sigma_2 \in \mathsf{Strat}_2} (\mathsf{Cost}(\mathsf{Outcome}(s, \sigma_1, \sigma_2))).$$

A strategy $\sigma_1 \in \mathsf{Strat}_1$ is said to be optimal from s if $\mathsf{Cost}_{\mathcal{G}}(s, \sigma_1) = \mathsf{OptCost}_{\mathcal{G}}(s)$. Since the optimal strategies may not always exist [5] we define ϵ optimal strategies. For $\epsilon > 0$ a strategy $\sigma_\epsilon \in \mathsf{Strat}_1$ is called ϵ-optimal if $\mathsf{OptCost}_{\mathcal{G}}(s) \leq \mathsf{Cost}_{\mathcal{G}}(s, \sigma_\epsilon) < \mathsf{OptCost}_{\mathcal{G}}(s) + \epsilon$. Given a PTG $\mathcal{G}$ and a bound $K \in \mathbb{Z}$, the *cost-optimal* reachability problem for PTGs is to decide whether there exists a strategy for player 1 such that $\mathsf{OptCost}_{\mathcal{G}}(s) \leq K$ from some starting state s.

▶ **Theorem 2.3** ([3]). *Cost-optimal reachability problem is undecidable for PTGs with* 3 *clocks.*

▶ **Theorem 2.4** ([5, 11, 13]). *The ϵ-optimal strategy is computable for* 1 *clock PTGs.*

3 Robust Semantics

Under the robust semantics of priced timed games the environment player – also called as the perturbator – is more privileged as it has the power to perturb any delay chosen by the controller by an amount in $[-\delta, \delta]$, where $\delta > 0$ is a pre-defined bounded quantity. However, in order to ensure time-divergence there is a restriction that the time delay at all locations of the RPTG must be $\geq \delta$. There are the following two perturbation semantics as defined in [7].

- *Excess semantics.* At any controller location, the time delay t chosen by the controller is altered to some $t' \in [t - \delta, t + \delta]$ by the perturbator. However, the constraints on the outgoing transitions of the controller locations are evaluated with respect to the time elapse t chosen by the controller. If the constraint is satisfied with respect to t, then the values of all variables which are not reset on the transition are updated with respect to t'; the variables which are reset obtain value 0.
- *Conservative semantics.* In this, the constraints on the outgoing transitions are evaluated with respect to t'.

In both semantics, the delays chosen by perturbator at his locations are not altered, and the constraints on outgoing transitions are evaluated in the usual way, as in PTG.

A Robust-Priced Timed Automata (RPTA) is an RPTG which has only controller locations. At all these locations, for any time delay t chosen by controller, perturbator can implicitely perturb t by a quantity in $[-\delta, \delta]$. The excess as well as the conservative perturbation semantics for RPTA are defined in the same way as in the RPTG. Note that our RPTA coincides with that of [7] when the cost functions at all target locations are of the form $cf : \mathbb{R}^n_{\geq 0} \to \{0\}$. Our RPTG are turn-based, and have cost funtions at the targets, while RPTGs studied in [7] are concurrent.

▶ **Definition 3.1** (Excess Perturbation Semantics). Let $\mathcal{R} = (L_1, L_2, L_{init}, \mathcal{C}, X, \eta, T, f_{goal})$ be a RPTG. Given a $\delta > 0$, the excess perturbation semantics of RPTG $\mathcal{R}$ is a LTS $[\![\mathcal{R}]\!] = (\mathcal{S}, A, E)$ where $\mathcal{S} = S_1 \cup S_2 \cup (T \times \mathbb{R}_{\geq 0})$, $A = A_1 \cup A_2$ and $E = E_1 \cup E_2$. We define the set of states, actions and transitions for each player below.

- $S_1 = L_1 \times \mathbb{R}^{|\mathcal{C}|}$ are the controller states,

- $S_2 = (L_2 \times \mathbb{R}^{|\mathcal{C}|}) \cup (S_1 \times \mathbb{R}_{\geq 0} \times X)$ are the perturbator states. The first kind of states are encountered at perturbator locations. The second kind of states are encountered when controller chooses a delay $t \in \mathbb{R}_{\geq 0}$ and a transition $e \in X$ at a controller location.
- $A_1 = \mathbb{R}_{\geq 0} \times X$ are controller actions
- $A_2 = (\mathbb{R}_{\geq 0} \times X) \cup [-\delta, \delta]$ are perturbator actions. The first kind of actions $(\mathbb{R}_{\geq 0} \times X)$ are chosen at states of the form $L_2 \times \mathbb{R}^{|\mathcal{C}|} \in S_2$, while the second kind of actions are chosen at states of the form $S_1 \times \mathbb{R}_{\geq 0} \times X \in S_2$,
- $E_1 = (S_1 \times A_1 \times S_2)$ is the set of controller transitions such that for a controller state (l, ν) and a controller action (t, e), $E_1((l, \nu), (t, e))$ is defined iff there is a transition $e = (l, g, a, r, l')$ in $\mathcal{R}$ such that $\nu + t \in [\![g]\!]$.
- $E_2 = S_2 \times A_2 \times (S_1 \cup S_2 \cup (T \times \mathbb{R}_{\geq 0}))$ is the set of perturbator transitions such that
  - For a perturbator state of the type (l, ν) and a perturbator action (t, e), we have $(l', \nu') = E_2((l, \nu), (t, e))$ iff there is a transition $e = (l, g, a, r, l')$ in $\mathcal{R}$ such that $\nu + t \in [\![g]\!]$, $\nu' = (\nu + t)[r := 0]$,
  - For a perturbator state of type $((l, \nu), t, e)$ and a perturbator action $\varepsilon \in [-\delta, \delta]$, we have $(l', \nu') = E_2(((l, \nu), t, e), \varepsilon)$ iff $e = (l, g, a, r, l')$, and $\nu' = (\nu + t + \varepsilon)[r := 0]$.

We now define the cost of the transitions, denoted as $\mathsf{Cost}(t, e)$ as follows :

- For controller transitions : $(l, \nu) \xrightarrow{(t,e)} ((l, \nu), t, e)$: the cost accumulated is $\mathsf{Cost}(t, e) = 0$.
- For perturbator transitions :
  - From perturbator states of type (l, ν) : $(l, \nu) \xrightarrow{t,e} (l', \nu')$, the cost accumulated is $\mathsf{Cost}(t, e) = t * \eta(l)$.
  - From perturbator states of type $((l, \nu), t, e)$: $((l, \nu), t, e) \xrightarrow{\varepsilon} (l', \nu')$, the cost accumulated is $(t + \varepsilon) * \eta(l)$. Note that although this transition has no edge choice involved and the perturbation delay chosen is $\varepsilon \in [-\delta, \delta]$, the controller action (t, e) chosen in the state (l, ν) comes into effect in this transition. Hence for the sake of uniformity, we denote the cost accumulated in this transition to be $\mathsf{Cost}(t + \varepsilon, e) = (t + \varepsilon) * \eta(l)$.

Note that we check satisfiability of the constraint g before the perturbation; however, the reset occurs after the perturbation. The notions of a path and a winning play are the same as in PTG. We shall now adapt the definitions of cost of a play, and a strategy for the excess perturbation semantics. Let $\rho = \langle s_1, (t_1, e_1), s_2, (t_2, e_2), \cdots (t_{n-1}, e_{n-1}), s_n \rangle$ be a path in the LTS $[\![\mathcal{R}]\!]$. Given a $\delta > 0$, for a finite play ρ ending in target location, we define $\mathsf{Cost}^{\delta}_{\mathcal{R}}(\rho) = \sum_{i=1}^{n} \mathsf{Cost}(t_i, e_i) + f_{goal}(l_n)(\nu_n)$ as the sum of the costs of all transitions as defined above along with the value from the cost function of the target location l_n. Also, we re-define the cost of a strategy σ_1 from a state s for a given $\delta > 0$ as $\mathsf{Cost}^{\delta}_{\mathcal{R}}(s, \sigma_1) = \sup_{\sigma_2 \in \mathsf{Strat}_2(\mathcal{R})} \mathsf{Cost}^{\delta}_{\mathcal{R}}(\mathsf{Outcome}(s, \sigma_1, \sigma_2))$. Similarly, $\mathsf{OptCost}^{\delta}_{\mathcal{R}}$ is the optimal cost under excess perturbation semantics for a given $\delta > 0$ defined as

$$\mathsf{OptCost}^{\delta}_{\mathcal{R}}(s) = \inf_{\sigma_1 \in \mathsf{Strat}_1(\mathcal{R})} \sup_{\sigma_2 \in \mathsf{Strat}_2(\mathcal{R})} (\mathsf{Cost}^{\delta}_{\mathcal{R}}(\mathsf{Outcome}(s, \sigma_1, \sigma_2))).$$

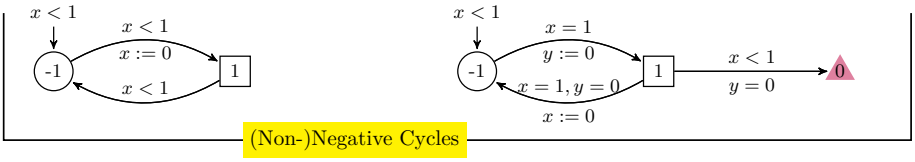
Since optimal strategies may not always exist, we define $\epsilon-$optimal strategies such that for every $\epsilon > 0$, $\mathsf{OptCost}^{\delta}_{\mathcal{R}}(s) \leq \mathsf{Cost}^{\delta}_{\mathcal{R}}(s, \sigma_1) < \mathsf{OptCost}^{\delta}_{\mathcal{R}}(s) + \epsilon$. Given a δ and a RPTG $\mathcal{R}$ with a single clock x, a strategy σ_1 is called $(\epsilon, N)-$*acceptable* [5] for $\epsilon > 0, N \in \mathbb{N}$ when (1)it is memoryless, (2)it is $\epsilon-$optimal and (3)there exist N consecutive intervals $(I_i)_{1 \leq i \leq N}$ partitioning $[0, 1]$ such that for every location l, for every $1 \leq i \leq N$ and every integer $\alpha < M$ (where M is the maximum bound on the clock value), the function that maps the clock values $\nu(x)$ to the cost of the strategy σ_1 at every state $(l, \nu(x))$, $(\nu(x) \mapsto \mathsf{Cost}^{\delta}_{\mathcal{R}}((l, \nu(x)), \sigma_1))$ is

affine for every interval $\alpha + I_i$. Also, the strategy σ_1 is constant over the values $\alpha + I_i$ at all locations, that is, when $\nu(x) \in \alpha + I_i$, the strategy $\sigma_1(l, \nu(x))$ is constant. The number N is an important attribute of the strategy as it establishes that the strategy does not fluctuate infinitely often and is implementable.

Now, we shall define limit variations of costs, strategies and values as $\delta \to 0$. The *limit-cost* of a controller strategy σ_1 from state s is defined over all plays ρ starting from s that are compatible with σ_1 as:

$$\mathsf{LimCost}_{\mathcal{R}}(s, \sigma_1) = \lim_{\delta \to 0} \sup_{\sigma_2 \in \mathsf{Strat}_2(\mathcal{R})} \mathsf{Cost}^{\delta}_{\mathcal{R}}(\mathsf{Outcome}(s, \sigma_1, \sigma_2)).$$

The *limit strategy upper-bound problem* [7] for excess perturbation semantics asks, given a RPTG $\mathcal{R}$, state $s = (l, \mathbf{0})$ with cost 0 and a rational number K, whether there exists a strategy σ_1 such that $\mathsf{LimCost}_{\mathcal{R}}(s, \sigma_1) \leq K$. The following are the main results of [7].



▶ **Theorem 3.2** (Known results [7]).

1. *The limit-strategy upper-bound problem is undecidable for RPTA and RPTG under excess perturbation semantics, for* ≥ 10 *clocks.*
2. *For a fixed* $\delta \in [0, \frac{1}{3}]$*, and a given RPTA* $\mathcal{A}$*, a target location* l *and a rational* K*, it is undecidable whether* $\inf_{\sigma_1} \sup_{\sigma_2} cost_{\sigma_1,\sigma_2}(\rho) < K$ *such that* ρ *ends in* l*.* $cost_{\sigma_1,\sigma_2}(\rho)$ *is the cost of the unique run* ρ *obtained from the pair of strategies* (σ_1, σ_2)*.*

We consider a semantic subclass of RPTGs in which the accumulated cost of any cycle is non-negative: that is, any iteration of a cycle will always have a non-negative cost. Consider the two cycles depicted. The one on top has a non-negative cost, while the one below always has a negative cost. In the cycle below, the perturbator will not perturb, since that will lead to a target state. In the rest of the paper, we consider this semantic class of RPTGs (RPTAs), and prove decidability and undecidability results; however, we will refer to them as RPTGs(RPTAs). Our key contributions are the following theorems.

▶ **Theorem 3.3.** *The limit-strategy upper-bound problem is undecidable for RPTA with 5 clocks, location prices in* $\{0, 1\}$*, and cost functions* $cf : \mathbb{R}^n_{\geq 0} \to \{0\}$ *at all target locations.*

▶ **Theorem 3.4.** *Given a 1-clock RPTG* $\mathcal{R}$ *and a* $\delta > 0$*, we can compute* $\mathsf{OptCost}^{\delta}_{\mathcal{R}}(s)$ *for every state* $s = (l, \nu)$*. For every* $\epsilon > 0$*, there exists an* $N \in \mathbb{N}$ *such that the controller has an* (ϵ, N)*-acceptable strategy.*

The rest of the paper is devoted to the proof sketches of these two theorems, while we give detailed proofs in [10].

4 Undecidability with 5 clocks

In this section, we improve the result of [7] by showing that the limit strategy upper bound problem is undecidable for robust priced timed automata with 5 or more clocks. The undecidability result is obtained using a reduction to the halting problem of two-counter machines.

A two-counter machine has counters C_1 and C_2, and a list of instructions $I_1, I_2, \ldots, I_n$, where I_n is the *halt instruction*. For each $1 \leq i \leq n-1$, I_i is one of the following instructions: `increment` c_b`:` $c_b := c_b + 1$; *goto* I_j, for $b = 1$ or 2, `decrement` c_b `with zero test:` *if* $(c_b = 0)$ *goto* I_j *else* $c_b := c_b - 1$; *goto* I_j, where c_1, c_2 represent the counter values. The initial values of both counters are 0. Given the initial configuration $(I_1, 0, 0)$ the halting problem for two counter machines is to find if the configuration (I_n, c_1, c_2) is reachable, with $c_1, c_2 \geq 0$. This problem is known to be undecidable.

We simulate the two counter machine using a RPTA with 5 clocks x_1, z, x_2, y_1 and y_2 under the excess perturbation semantics. The counters are encoded in clocks x_1 and z as $x_1 = \frac{1}{2^i} + \varepsilon_1$ and $z = \frac{1}{2^j} + \varepsilon_2$ where i, j are respectively the values of counters C_1, C_2, and ε_1 and ε_2 denote accumulated values due to possible perturbations. Clocks x_2, y_1 and y_2 help with the rough work. The simulation is achieved as follows: for each instruction, we have a module simulating it. Upon entering the module, the clocks are in their normal form i.e. $x_1 = \frac{1}{2^i} + \varepsilon_1, z = \frac{1}{2^j} + \varepsilon_2$ and $x_2 = 0$ and $y_1 = y_2 = 0$.

4.1 Increment module

The module in Figure 1 simulates the increment of counter C_1. The value of counter C_2 remains unchanged since the value of clock z remains unchanged at the exit from the module. Upon entering A the clock values are $x_1 = \frac{1}{2^i} + \varepsilon_1, z = \frac{1}{2^j} + \varepsilon_2, x_2 = y_1 = y_2 = 0$. Here ε_1 and ε_2 respectively denote the perturbations accumulated so far. We denote by α, the value of clock x_1, i.e. $\frac{1}{2^i} + \varepsilon_1$. Thus at A, the delay is $1 - \alpha$. Note that the dashed edges are unperturbed (this is a short hand notation. A small gadget that implements this is described in [10], so $x_1 = 1$ on entering B. No time elapse happens at B, and at C, controller chooses a delay t. This t must be $\frac{\alpha}{2}$ to simulate the increment correctly. t can be perturbed by an amount δ by the perurbator, where δ can be both positive or negative, obtaining $x_2 = t + \delta, x_1 = 0, y_1 = 1 - \alpha + t + \delta$ on entering D. At D, the delay is $\alpha - t - \delta$. Thus the total delay from the entry point A in this module to the mChoice module is 1 time unit. At the entry of the *mChoice* (*mChoice* and Restore modules are in [10]) module, the clock values are $x_1 = \alpha - t - \delta, z = 1 + \frac{1}{2^j} + \varepsilon_2, x_2 = \alpha, y_1 = 1, y_2 = 0$. To correctly simulate the increment of C_1, t should be exactly $\frac{\alpha}{2}$.

At the mChoice module, perturbator can either continue the simulation by going through the Restore module or verify the correctness of controller's delay (check $t = \frac{\alpha}{2}$). The mChoice module adds 3 units to the values of x_1, x_2 and z, and resets y_1, y_2. Due to the mChoice module, the clock values are $x_1 = 3 + \alpha - t - \delta, z = 4 + \frac{1}{2^j} + \varepsilon_2, x_2 = 3 + \alpha, y_1 = 1, y_2 = 0$. If perturbator chooses to continue the simulation, then Restore module brings all the clocks back to normal form. Hence upon entering F, the clock values are $x_1 = \alpha - t - \delta, z = \frac{1}{2^j} + \varepsilon_2, x_2 = y_1 = 1, y_2 = 0$. This value of x_1 is $\frac{\alpha}{2} + \varepsilon_1$, since $t = \frac{\alpha}{2}$ and $\varepsilon_1 = -\delta$, the perturbation effect.

Let us now see how perturbator verifies $t = \frac{\alpha}{2}$ by entering the Choice module. The Choice module also adds 3 units to the values of x_1, x_2 and z, and resets y_1, y_2. The module $Test\ Inc_{>}^{C_1}$ is invoked to check if $t > \frac{\alpha}{2}$, and the module $Test\ Inc_{<}^{C_1}$ is invoked to check if $t < \frac{\alpha}{2}$. Note that using the mChoice module and the Choice module one after the other, the clock values upon entering $Test\ Inc_{>}^{C_1}$ or $Test\ Inc_{<}^{C_1}$ are $x_1 = 6 + \alpha - t - \delta, z = 7 + \frac{1}{2^j} + \varepsilon_2, x_2 = 6 + \alpha, y_1 = 0, y_2 = 0$.

Test $\mathbf{Inc_{>}^{C_1}}$: The delay at A' is $1 - \alpha + t + \delta$, obtaining $x_2 = 7 + t + \delta$, and the cost accumulated is $1 - \alpha + t + \delta$. At B', $1 - t - \delta$ time is spent, obtaining $x_1 = 1 - t - \delta$. Finally, at C', a time $t + \delta$ is spent, and at D', one time unit, making the total cost accumulated $2 - \alpha + 2t + 2\delta$ at the target location. The cost function at the target assigns the cost 0 for

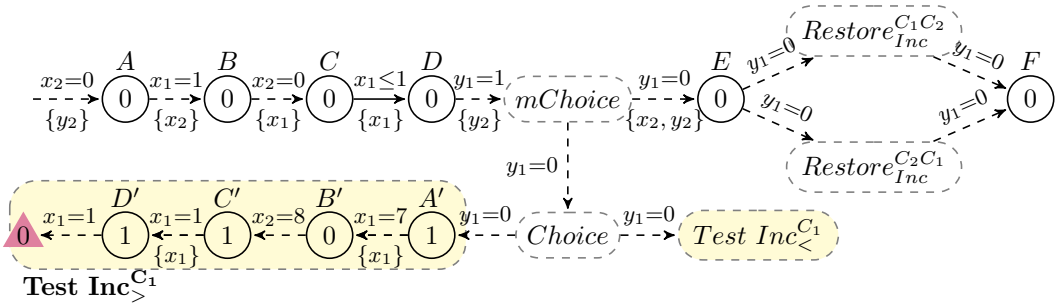


Figure 1 Increment C_1 module : The module keeps the fractional part of the clock z unchanged. The dashed edges represent unperturbed edges (detailed in [10]).

all valuations, hence the total cost to reach the target is $2 + 2t - \alpha + 2\delta$ which is greater than $2 + 2\delta$ iff $2t - \alpha > 0$, i.e. iff $t > \frac{\alpha}{2}$.

▶ **Lemma 4.1.** *Assume that an increment C_b ($b \in \{0, 1\}$) module is entered with the clock valuations in their normal forms. Then controller has a strategy to reach either location l_j corresponding to instruction I_j of the two-counter machine or a target location is reached with cost at most $2 + |2\delta|$, where δ is the perturbation added by perturbator.*

4.2 Complete Reduction

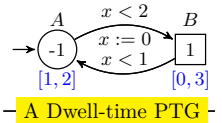
The entire reduction consists of constructing a module corresponding to each instruction I_i, $1 \leq i \leq n$, of the two-counter machine. The first location of the module corresponding to instruction I_1 is the initial location. We simulate the halting instruction I_n by a target location with cost function $cf : \mathbb{R}^5_{\geq 0} \to \{0\}$. We denote the robust timed automaton simulating the two counter machine by $\mathcal{A}$, s is the initial state $(l, \mathbf{0}, \mathbf{0})$.

▶ **Lemma 4.2.** *The two counter machine halts if and only if there is a strategy σ of controller such that $limcost_{\mathcal{A}}(\sigma, s) \leq 2$.*

The details of the decrement and zero test modules are in [10]. They are similar to the increment module; if player 2 desires to verify the correctness of player 1's simulation, a cost $> 2 + |2\delta|$ is accumulated on reaching a target location iff player 1 cheats. In the limit, as $\delta \to 0$, the limcost will be > 2 iff controller cheats. The other possibility to obtain a limcost > 2 is when the two counter machine does not halt.

5 Decidability of One-clock RPTG

In order to show the decidability of the optimal reachability game for 1 clock RPTG $\mathcal{R}$ and a fixed $\delta > 0$, we perform a series of reachability and optimal cost preserving transformations. The idea is to reduce the RPTG into a simpler priced timed game, while preserving the optimal costs. The advantages of this conversion is that the semantics of PTGs are easier to understand, and one could adapt known algorithms to solve PTGs. On the other hand, the PTGs that we obtain are 1-clock PTGs with dwell-time requirement (having restrictions on minimum as well as maximum amount of time spent at certain locations), see for example, a

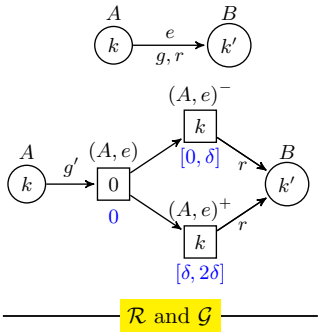


dwell-time PTG with two locations A, B. A minimum of 1 and a maximum of two units of time should be spent at A, while a maximum of 3 time units can be spent at B. If we wish to model this using standard PTGs, we need one extra clock and we can not use the decidability results of 1 clock PTG to show the decidability of our model. We show in Section 5.4 how to solve 1-clock PTGs with dwell-time requirements.

Our transformations are as follows: (i) for a given δ, our first transformation reduces the RPTG $\mathcal{R}$ into a dwell-time PTG $\mathcal{G}$ (Section 5.1); (ii) our second transformation restricts to dwell-time PTGs where the clock is bounded by $1+\delta$. To achieve this, we use a notion of *fractional resets*, and denote these PTGs as $\mathcal{G}_{\mathcal{F}}$ (Section 5.2); (iii) our third and last transformation restricts $\mathcal{G}_{\mathcal{F}}$ without resets (Section 5.3). The reset-free dwell-time PTG is denoted ${\mathcal{G}_{\mathcal{F}}}'$. For each transformation, we prove that the optimal cost in each state of the original game is the same as the optimal cost at some corresponding state of the new game. We also show that an (ϵ, N)-strategy of the original game can be computed from some (ϵ', N')-strategy in the new game. The details of each transformation and correctness is established in subequent sections. We then solve ${\mathcal{G}_{\mathcal{F}}}'$ employing a technique inspired by [5] while ensuring that the robust semantics are satisfied.

5.1 Transformation 1: RPTG $\mathcal{R}$ to dwell-time PTG $\mathcal{G}$

Given a one clock RPTG $\mathcal{R} = (L_1, L_2, \{x\}, X, \eta, T, f_{goal})$ and a $\delta > 0$, we construct a dwell-time PTG $\mathcal{G} = (L_1, L_2 \cup L', \{x\}, X', \eta', T, f_{goal})$. All the controller, perturbator locations of $\mathcal{R}$ (L_1 and L_2) are carried over respectively as player 1, player 2 locations in $\mathcal{G}$. In addition, we have some new player 2 locations L' in $\mathcal{G}$. The dwell-time PTG $\mathcal{G}$ constructed has dwell-time restrictions for the new player 2 locations L'. The locations of L' are either urgent, or have a a dwell-time of $[\delta, 2\delta]$ or $[0, \delta]$. All the perturbator transitions of $\mathcal{R}$ are retained as it is in $\mathcal{G}$. Every transition in $\mathcal{R}$ from a controller location A to some location B is replaced in $\mathcal{G}$ by a game graph as shown. Let $e = (A, g, r, B)$ be the transition from a controller location A to a location B with guard g, and reset r. Depending on the guard g, in the transformed game graph, we have the new guard g'. If g is $x = H$, then g' is $x = H - \delta$, while if g is $H < x < H + 1$, then g' is $H - \delta < x < H + 1 - \delta$, for $H > 0$. When g is $0 < x < K$, then g' is $0 \leq x < K - \delta$ and $x = 0$ stays unchanged. It can be seen that doing this transformation to all the controller edges of a RPTG $\mathcal{R}$ gives rise to a dwell-time PTG $\mathcal{G}$.



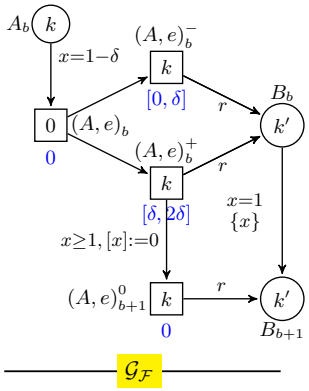
Lets consider the transition from A to B in $\mathcal{R}$. Assume that the transition from A to B (called edge e) had a constraint $x = 1$, and assume that $x = \nu$ on entering A. Then, in $\mathcal{R}$, controller elapses a time $1 - \nu$, and reaches B; however on reaching B, the value of x is in the range $[1 - \delta, 1 + \delta]$ depending on the perturbation. Also, the cost accumulated at A is $k * (1 - \nu + \gamma)$, where $\gamma \in [-\delta, \delta]$. To take into consideration these semantic restrictions of $\mathcal{R}$, we transform the RPTG $\mathcal{R}$ into a dwell-time PTG $\mathcal{G}$. First of all, we change the constraint $x = 1$ into $x = 1 - \delta$ from A (a player 1 location) and enter a new player 2 location (A, e). This player 2 location is an urgent location. The correct strategy for player 1 is to spend a time $1 - \nu - \delta$ at A (corresponding to the time $1 - \nu$ he spent at A in $\mathcal{R}$). At (A, e), player 2 can either proceed to one of the player 2 locations $(A, e)^-$ or $(A, e)^+$. The player 2 location (A, e) models perturbator's choices of positive or negative perturbation in $\mathcal{R}$. If player 2 goes to $(A, e)^-$, then on reaching B, the value of x is in the interval $[1 - \delta, 1]$ (this

corresponds to perturbator's choice of $[-\delta, 0]$ in $\mathcal{R}$) and if he goes to $(A,e)^+$, then the value of x at B is in the interval $[1, 1+\delta]$ (this corresponds to perturbator's choice of $[0,\delta]$ in $\mathcal{R}$). The reset happening in the transition from A to B in $\mathcal{R}$ is now done on the transition from $(A,e)^-$ to B and from $(A,e)^+$ to B. Thus, note that the possible ranges of x as well as the accumulated cost in $\mathcal{R}$ while reaching B are preserved in the transformed dwell-time PTG.

▶ **Lemma 5.1.** *Let $\mathcal{R}$ be a RPTG and $\mathcal{G}$ be the corresponding dwell-time PTG obtained using the transformation above. Then for every state s in $\mathcal{R}$, $\mathsf{OptCost}_{\mathcal{R}}(s) = \mathsf{OptCost}_{\mathcal{G}}(s)$. An $(\epsilon, N)-$strategy in $\mathcal{R}$ can be computed from a $(\epsilon, N)-$strategy in $\mathcal{G}$ and vice versa.*

Proof in [10].

5.2 Transformation 2: Dwell-time PTG $\mathcal{G}$ to Dwell-time FRPTG $\mathcal{G}_{\mathcal{F}}$



Recall that the locations of the dwell-time PTG $\mathcal{G}$ is $L_1 \cup L_2 \cup L'$ where $L_1 \cup L_2$ are the set of locations of $\mathcal{R}$, and L' are new player 2 locations introduced in $\mathcal{G}$. In this section, we transform the dwell-time PTG $\mathcal{G}$ into a dwell-time PTG $\mathcal{G}_{\mathcal{F}}$ having the restriction that the value of x is in [0,1] at all locations corresponding to $L_1 \cup L_2$, and is in $[0, 1+\delta]$ at all locations corresponding to L'. While this transformation is the same as that used in [5], the main difference is that we introduce special resets called *fractional resets* which reset only the integral part of clock x while its fractional part is retained. For instance, if the value of x was 1.3, then the operation $[x] := 0$ makes the value of x to be 0.3. Given a one clock, dwell-time PTG $\mathcal{G} = (L_1, L_2 \cup L', \{x\}, X, \eta, T, f_{goals})$ with M being the maximum value that can be assumed by clock x, we define a dwell-time PTG with fractional resets (FRPTG) $\mathcal{G}_{\mathcal{F}}$. In $\mathcal{G}_{\mathcal{F}}$, we have $M+1$ copies of the locations in $L_1 \cup L_2$ as well as the locations in L' with dwell time $[0,\delta]$, $[0,0]$. These $M+1$ copies of L' have the same dwell-time restrictions in $\mathcal{G}_{\mathcal{F}}$. The copies are indexed by $i, 0 \leq i \leq M$, capturing the integral part of clock x in $\mathcal{G}$. Finally, we have in $\mathcal{G}$, the locations of L' with dwell-time restriction $[\delta, 2\delta]$. For each such location $(A,e)^+$, we have in $\mathcal{G}_{\mathcal{F}}$, the locations $(A,e)_i^+$ and $(A,e)_{i+1}^0$ for $0 \leq i \leq M$. The dwell-time restriction for $(A,e)_i^+$ is same as $(A,e)^+$, while locations $(A,e)_{i+1}^0$ are urgent. The prices of locations are carried over as they are in the various copies.

The transitions in $\mathcal{G}_{\mathcal{F}}$ consists of the following: (1) $l_i \xrightarrow{(g-i)\cap 0\leq x<1} m_i$[1] if $l \xrightarrow{g} m \in X$; (2) $l_i \xrightarrow{(g-i)\cap 0\leq x<1;\{x\}} m_0$ if $l \xrightarrow{g;\{x\}} m \in X$; (3) $l_i \xrightarrow{x=1,\{x\}} l_{i+1}$, for $l \in L_1 \cup L_2$, and $(A,e)_i^+ \xrightarrow{x\geq 1,[x]:=0} (A,e)_{i+1}^0$ for $i < M$. Consider for example, the constraint g' between A and (A,e) as $x = (b+1) - \delta$ in $\mathcal{G}$. Then the value of x is $b + (1-\delta)$ for $b < M$ when $(A,e)^+$ is entered in $\mathcal{G}$. The location $(A,e)^+$ with $\nu(x) = b + (1-\delta)$ is represented in $\mathcal{G}_{\mathcal{F}}$ as $(A,e)_b^+$ with $\nu(x) = 1 - \delta$. If player 2 spends $[\delta, 2\delta]$ time at $(A,e)^+$ in $\mathcal{G}$, then $\nu(x) \in [b+1, b+1+\delta]$. If there are no resets to goto B, then $\nu(x) \in [b+1, (b+1)+\delta]$ at B. Correspondingly in $\mathcal{G}_{\mathcal{F}}$, $\nu(x) \in [1, 1+\delta]$ at $(A,e)_b^+$. By construction, B_b is not reachable, since we check $0 \leq x < 1$ on the transition to B_b. The fractional reset is employed to obtain $x = \delta$ while moving to $(A,e)_{b+1}^0$. This ensures that $x = \delta$ on reaching B_{b+1}, thereby

[1] $g - i$ represents the constraint obtained by shifting the constraint by $-i$

preserving the perturbation, and keeping $x < 1$. A normal reset would have destroyed the value obtained by perturbation. The mapping f between states of $\mathcal{G}$ and $\mathcal{G}_{\mathcal{F}}$ is as follows: $f(l, x) = (l_b, x - b)$, $b < M$, and $x \in [b, b + 1]$, $l \in L_1 \cup L_2$, $f((A, e), x) = ((A, e)_b, x - b)$, $b < M$, and $x \in [b, b + 1]$, $f((A, e)^-, x) = ((A, e)_b^-, x - b)$, $b < M$, and $x \in [b, b + 1]$. Finally, $f((A, e)^+, x) = ((A, e)_b^+, x - b)$, $b < M$, and $x \in [b, b + 1] \cup [b + 1, b + 2]$. Note that in the last case, the value of $x - b$ can exceed 1 but is less than or equal to $1 + \delta$.

▶ **Lemma 5.2.** *For every state (l, ν) in $\mathcal{G}$, $\mathsf{OptCost}_{\mathcal{G}}(l, \nu)$ in $\mathcal{G}$ is the same as $\mathsf{OptCost}_{\mathcal{G}_{\mathcal{F}}}(f(l, \nu))$ in $\mathcal{G}_{\mathcal{F}}$. For every $\epsilon > 0$, $N \in \mathbb{N}$, an (ϵ, N)-acceptable strategy in $\mathcal{G}$ can be computed from an (ϵ, N)-acceptable strategy in $\mathcal{G}_{\mathcal{F}}$ and vice versa.*

5.3 Transformation 3: Dwell-time FRPTG $\mathcal{G}_{\mathcal{F}}$ to resetfree FRPTG $\mathcal{G}_{\mathcal{F}}{}'$

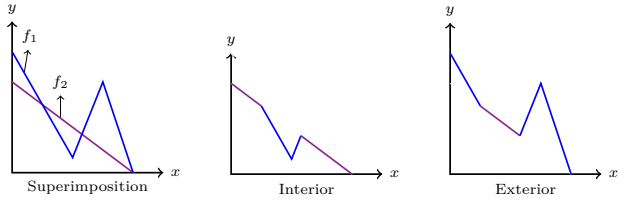
We now apply the final transformation to the FRPTG $\mathcal{G}_{\mathcal{F}}$ and construct a reset-free version of the FRPTG denoted $\mathcal{G}_{\mathcal{F}}{}'$. Assume that there are a total of n resets (including fractional resets) in the FRPTG. $\mathcal{G}_{\mathcal{F}}{}'$ consists of $n + 1$ copies of the FRPTG : $\mathcal{G}_{\mathcal{F}0}, \mathcal{G}_{\mathcal{F}1}, \ldots, \mathcal{G}_{\mathcal{F}n}$. Given the locations L of the FRPTG, the locations of $\mathcal{G}_{\mathcal{F}i}$ are L^i, $0 \leq i \leq n$. $\mathcal{G}_{\mathcal{F}0}$ starts with l^0, where l is the initial location of the FRPTG and continues until a resetting transition happens. At the first resetting transition, $\mathcal{G}_{\mathcal{F}0}$ makes a transition to $\mathcal{G}_{\mathcal{F}1}$. The nth copy is directed to a sink target location S with cost function $cf : \mathbb{R}_{\geq 0} \to \{+\infty\}$ on the $(n + 1)$th reset. Note that each $\mathcal{G}_{\mathcal{F}i}$ is reset-free. One crucial property of each $\mathcal{G}_{\mathcal{F}i}$ is that on entering with some value of x in $[0, \delta]$, the value of x only increases as the transitions go along in $\mathcal{G}_{\mathcal{F}i}$; moreover, $x \leq 1 + \delta$ in each $\mathcal{G}_{\mathcal{F}i}$ by construction. The formal details and proof of Lemma 5.3 can be found in [10]. Using the cost function of S and those of the targets, we compute the optimal cost functions for all the locations of the deepest component $\mathcal{G}_{\mathcal{F}n}$. The cost functions of the locations of $\mathcal{G}_{\mathcal{F}i}$ are used to compute that of $\mathcal{G}_{\mathcal{F}i-1}$, and so on until the cost function of l^0, the starting location of $\mathcal{G}_{\mathcal{F}0}$ is computed. An example can be seen in [10].

▶ **Lemma 5.3.** *For every state (l, ν) in $\mathcal{G}_{\mathcal{F}}$, $\mathsf{OptCost}_{\mathcal{G}_{\mathcal{F}}}(l, \nu) = \mathsf{OptCost}_{\mathcal{G}_{\mathcal{F}}{}'}(l^0, \nu)$, where $\mathcal{G}_{\mathcal{F}}{}'$ is the resetfree* FRPTG*. For every $\epsilon > 0, N \in \mathbb{N}$, given an (ϵ, N)-acceptable strategy σ' in $\mathcal{G}_{\mathcal{F}}{}'$, we can compute a $(2\epsilon, N)$-acceptable strategy σ in $\mathcal{G}_{\mathcal{F}}$ and vice versa.*

5.4 Solving the Resetfree FRPTG

Before we sketch the details, let us introduce some key notations. Observe that after our simplifying transformations, the cost functions cf are piecewise-affine continuous functions that assign a value to every valuation $x \in [0, 1 + \delta]$ (construction of FRPTG ensures $x \leq 1+\delta$ always). The *interior* of two cost functions f_1 and f_2 is a cost function $f_3 : [0, 1 + \delta] \to \mathbb{R}$ defined by $f_3(x) = \min(f_1(x), f_2(x))$. Similarly, the *exterior* of f_1 and f_2 is a cost function $f_4 : [0, 1 + \delta] \to \mathbb{R}$ defined as $f_4(x) = \max(f_1(x), f_2(x))$. Clearly, f_3 and f_4 are also piecewise-affine continuous. The interior and exterior can be easily computed by *superimposing* f_1 and f_2 as shown graphically in the example by computing lower envelope and upper envelope respectively.

We now work on the reset-free components $\mathcal{G}_{\mathcal{F}i}$, and give an algorithm to compute $OptCost_{\mathcal{G}_{\mathcal{F}i}}(l, \nu)$ for every state (l, ν) of $\mathcal{G}_{\mathcal{F}i}$, $\nu(x) \in [0, 1 + \delta]$. We also show the existence of an N such that for any $\epsilon > 0$, and every $l \in L^i, \nu(x) \in [0, 1 + \delta]$, an (ϵ, N)-acceptable strategy can be computed. Consider the location of $\mathcal{G}_{\mathcal{F}i}$ that has the smallest price and call it l_{min}. If this is a player 1 location, then intuitively, player 1 would want to spend as much time as possible here, and if this is a player 2 location, then player 2 would want to spend as less time as possible here. By our assumption, all the cycles in $\mathcal{G}_{\mathcal{F}i}$ are non-negative, and

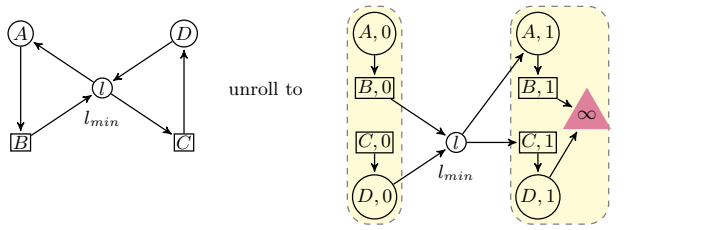


Example Illustrating SuperImposition, Interior and Exterior

hence if l_{min} is part of a cycle, revisiting it will only increase the total cost if at all. Player 1 thus would like to spend all the time he wants to during the first visit itself. We now prove that this is indeed the case. We consider two cases separately.

5.4.1 l_{min} is a Player 1 location

We split $\mathcal{G}_{\mathcal{F}i}$ such that l_{min} is visited only once. We transform $\mathcal{G}_{\mathcal{F}i}$ into $\mathcal{G}_{\mathcal{F}}{}''$ which has two copies of all locations except l_{min} such that corresponding to every location $l \neq l_{min}$, we have the copies $(l,0)$ and $(l,1)$. A special target location S is added with cost function assigning $+\infty$ to all clock valuations.



Duplicate $L - l_{min}$

Given the transitions X of $\mathcal{G}_{\mathcal{F}i}$, the FRPTG $\mathcal{G}_{\mathcal{F}}{}''$ has the following transitions.

- if $l \xrightarrow{g} l' \in X$ and $l, l' \neq l_{min}$ then $(l,0) \xrightarrow{g} (l',0)$ and $(l,1) \xrightarrow{g} (l',1)$
- if $l \xrightarrow{g} l' \in X$ and $l' = l_{min}$ then $(l,0) \xrightarrow{g} l_{min}$ and $(l,1) \xrightarrow{g} S$,
- if $l_{min} \xrightarrow{g} l$, then $l_{min} \xrightarrow{g} (l,1)$

▶ **Lemma 5.4.** *For every state (l,ν) if $\nu \in [0, 1+\delta]$ and $l \neq l_{min}$, we have that $\mathit{OptCost}_{\mathcal{G}_{\mathcal{F}i}}(l,\nu) = \mathit{OptCost}_{\mathcal{G}_{\mathcal{F}}{}''}((l,0),\nu)$ and $\mathit{OptCost}_{\mathcal{G}_{\mathcal{F}i}}(l_{min},\nu) = \mathit{OptCost}_{\mathcal{G}_{\mathcal{F}}{}''}(l_{min},\nu)$.*

We give an intuition for Lemma 5.4. Locations $(l,0)$ have all the transitions available to location l in $\mathcal{G}_{\mathcal{F}i}$. Also, any play in $\mathcal{G}_{\mathcal{F}}{}''$ which is compatible with a winning strategy of player 1 in $\mathcal{G}_{\mathcal{F}i}$ contains only one of the locations $(l,0),(l,1)$ by construction of $\mathcal{G}_{\mathcal{F}}{}''$. The outcomes from $(l,0)$ are more favourable than $(l,1)$ for l as a player 1 location. Based on these intuitions, we conclude that $\mathsf{OptCost}_{\mathcal{G}_{\mathcal{F}i}}(l,\nu)$ is same as that for $((l,0),\nu)$. This observation also leads to the $\epsilon-$optimal strategy being the same as that for $(l,0)$. Given a strategy σ' in $\mathcal{G}_{\mathcal{F}}{}''$, we construct σ in $\mathcal{G}_{\mathcal{F}i}$ as $\sigma(l,\nu) = \sigma'((l,0),\nu)$. Further, any strategy that revisits l_{min} in $\mathcal{G}_{\mathcal{F}i}$ cannot be winning for player 1, since all cycles are non-negative; we end up at S with cost ∞ in $\mathcal{G}_{\mathcal{F}}{}''$. However, all strategies that do not revisit l_{min} in $\mathcal{G}_{\mathcal{F}i}$ are preserved in $\mathcal{G}_{\mathcal{F}}{}''$, and hence $\mathsf{OptCost}_{\mathcal{G}_{\mathcal{F}i}}(l_{min},\nu) = \mathsf{OptCost}_{\mathcal{G}_{\mathcal{F}}{}''}(l_{min},\nu)$.

We iteratively solve the part of $\mathcal{G}_{\mathcal{F}}{}''$ with locations indexed 1 (i.e; $(l,1)$) in the same fashion (picking minimal price locations) each time obtaining a smaller PTG. Computing the cost function of the minimal price location of the last such PTG, and propagating this backward, we compute the cost function of l_{min}. We then use the cost function of l_{min} to solve the part of $\mathcal{G}_{\mathcal{F}}{}''$ with locations indexed 0 (i.e; $(l,0)$).

Algorithm 1: Optimal Cost Algorithm when l_{min} is a Player 1 location.

Let $l_1, \ldots, l_n$ be the successors of l_{min} with optcost functions $f_1, f_2 \cdots f_n$.;
STEP 1 : Superimpose : Superimpose all the optcost functions $f_1, f_2 \cdots f_n$.;
STEP 2 : Interior : Take the interior of the superimposition; call it f.;
Let f be composed of line segments $g_1, g_2 \cdots g_m$ such that $g_i \in \{f_1, \ldots, f_n\}$, for all i.
$\forall\ k$, let the domain of g_k be $[u_k, v_k]$. Set $i = m$.;
STEP 3 : Selective Replacement : **while** $i \geq 1$ **do**
  if *slope of* $g_i \leq -\eta(l_{min})$ **then**
    replace g_i with line h_i with slope $-\eta(l_{min})$ and passing through $(v_i, g_i(v_i))$;
    Let h_i intersect g_j (largest $j < i$) at some point $x = v_j''$, $v_j'' \in [u_j, v_j]$;
    Update domain of g_j from $[u_j, v_j]$ to $[u_j, v_j'']$;
    if $j < i - 1$ **then**
      Remove functions g_{j+1} to g_{i-1} from f
    Set $i = j$;
  else
    $i = i - 1$;
STEP 4 : Refresh Interior : Take the interior after STEP 3 and call it f'.;
if $l'' \to l_{min}$ **then**
  update the optcost function of l''

Computing the Optcost function of l_{min}: Algorithm 1 computes the optcost function for a player 1 location l_{min}, assuming all the constraints on outgoing transitions from l_{min} are the same, namely $x \in [0, 1]$. We discuss adapting the algorithm to work for transitions with different constraints in [10]. A few words on the notation used: if a location l has price $\eta(l)$, then slope associated with l is $-\eta(l)$ (see STEP 3 in Algorithm 1).

Let $l_1, \ldots, l_n$ be the successors of l_{min}, with cost functions $f_1, \ldots, f_n$. Each of these cost functions are piecewise affine continuous over the domain $[0, 1]$. The first thing to do is to superimpose $f_1, \ldots, f_n$, and obtain the cost function f corresponding to the interior of $f_1, \ldots, f_n$ (l_{min} is a player 1 location and would like to obtain the minimal cost, hence the interior). The line segments comprising f come from the various f_i. Let $dom(f) = [0, 1]$ be composed of $0 = u_{i_1} \leq v_{i_1} = u_{i_2} \leq \ldots u_{i_m} \leq v_{i_m} = 1$: that is, $f(x) = f_{i_j}(x)$, $dom(f_{i_j}) = [u_{i_j}, v_{i_j}]$, for $i_j \in \{1, 2, \ldots, n\}$ and $1 \leq j \leq m$. Let us denote f_{i_j} by g_j, for $1 \leq j \leq m$. Then, f is composed of $g_1, g_2, \ldots, g_m$, and $dom(f)$ is composed of $dom(g_1), \ldots, dom(g_m)$ from left to right. Let $dom(g_i) = [u_i, v_i]$. Step 2 of the algorithm achieves this.

For a given valuation $\nu(x)$, if l_{min} is an urgent location, then player 1 would go to a location l_k if the interior f is such that $f(\nu(x)) = g_k(\nu(x))$(the least cost is given by g_k, obtained from the outside cost function of l_k). If l_{min} is not an urgent location, then player 1 would prefer delaying t units at l_{min} so that $\nu(x) + t \in [u_i, v_i]$ rather than goto some location l_i if $g_i(\nu(x)) > \eta(l_{min})(v_i - \nu(x))$. Again, g_i is a part of the ouside cost function of l_i, and player 1 prefers delaying time at l_{min} rather than goto l_i since that minimizes the cost. In this case, the cost function f is refined by replacing the line segment g_i over $[u_i, v_i]$ by another line segment h_i passing through $(v_i, g_i(v_i))$, and having a slope $-\eta(l_{min})$. Step 3 of the algorithm does this.

Recall that by our transformation 2, the value of clock x in any player 1 location is $\leq 1 - \delta$. The value of x is in $[1 - \delta, 1 + \delta]$ only at a player 2 location $((A, e)_b)^+$ in the FRPTG, section 5.2). Hence, the domain of cost functions for player 1 locations is actually $[0, 1 - \delta]$,

and not $[0, 1+\delta]$. Let the domain of g_m be $[u_m, 1]$. Then we can split g_m into two functions g_m^1, g_m^2 with domains $[u_m, 1-\delta]$ and $[1-\delta, 1]$. Now, we ensure that no time is spent in the player 1 location l_{min} over $dom(g_m^2)$, by not applying step 3 of the algorithm for g_m^2. This way, selective replacement of the cost functions g_i occur only in the domain $[0, 1-\delta]$, and we remain faithful to transformation 2, and the semantics of RPTGs.

Computing Almost Optimal Strategies: The strategy corresponding to this computed optcost is derived as follows. f' is the optcost of location l_{min} computed in Step 4 of the algorithm. f' is composed of two kinds of functions (a) the functions g_i computed in step 2 as a result of the interior of superimposition and (b) functions h_i which replaced some functions g_j from f, corresponding to delay at l_{min}. For functions h_j of f' with domain $[u_j, v_j]$, we prescribe the strategy to delay at l_{min} till $x = v_j$ when entered with clock $x \in [u_j, v_j]$. For functions g_i, that come from f at Step 2, where g_i is part of some optcost function f_k, (f_k is the optcost function of one of the successors l_k of l_{min}), the strategy dictates moving immediately to l_k when entered with clock $x \in [u_i, v_i]$.

Termination: Finally, we prove the existence of a number N, the number of affine segments that appear in the cost functions of all locations. Start with the resetfree FRPTG with m locations having p segments in the outside cost functions. Let $\alpha(m, p)$ denote the total number of affine segments appearing in cost functions across all locations. The transformation of resetfree components $\mathcal{G}_{\mathcal{F}}$ into $\mathcal{G}_{\mathcal{F}}{}''$ gives rise to two smaller resetfree FRPTGs with $m-1$ locations each, after separating out l_{min}. The resetfree FRPTG $(\mathcal{G}_{\mathcal{F}}, 1)$ with $m-1$ locations indexed with 1 of the form $(l, 1)$ are solved first, these cost functions are added as outside cost functions to solve l_{min}, and finally, the cost function of l_{min} is added as an outside cost function to solve the resetfree FRPTG $(\mathcal{G}_{\mathcal{F}}, 0)$ with $m-1$ locations indexed with 0 of the form $(l, 0)$. Taking into account the new sink target location added, we have $\leq p+1$ segments in outside cost functions in $(\mathcal{G}_{\mathcal{F}}, 1)$. This gives atmost $\beta = \alpha(m-1, p+1)$ segments in solving $(\mathcal{G}_{\mathcal{F}}, 1)$, and $\alpha(1, p+\beta) = \gamma$ segments to solve l_{min}, and finally $\alpha(m-1, p+\gamma)$ segments to solve $(\mathcal{G}_{\mathcal{F}}, 0)$. Solving this, one can easily check that $\alpha(m, p)$ is atmost triply exponential in the number of locations m of the resetfree component $\mathcal{G}_{\mathcal{F}}$. Obtaining a bound of the number of affine segments, it is easy to see that Algorithm 1 terminates; the time taken to compute almost optimal strategies and optcost functions is triply exponential.

We illustrate the computation of Optcost of a Player 1 location in Figure 2. The proof of Lemma 5.5 is given in [10], while Lemma 5.6 follows from Lemma 5.5 and Step 4 of Algorithm 1.

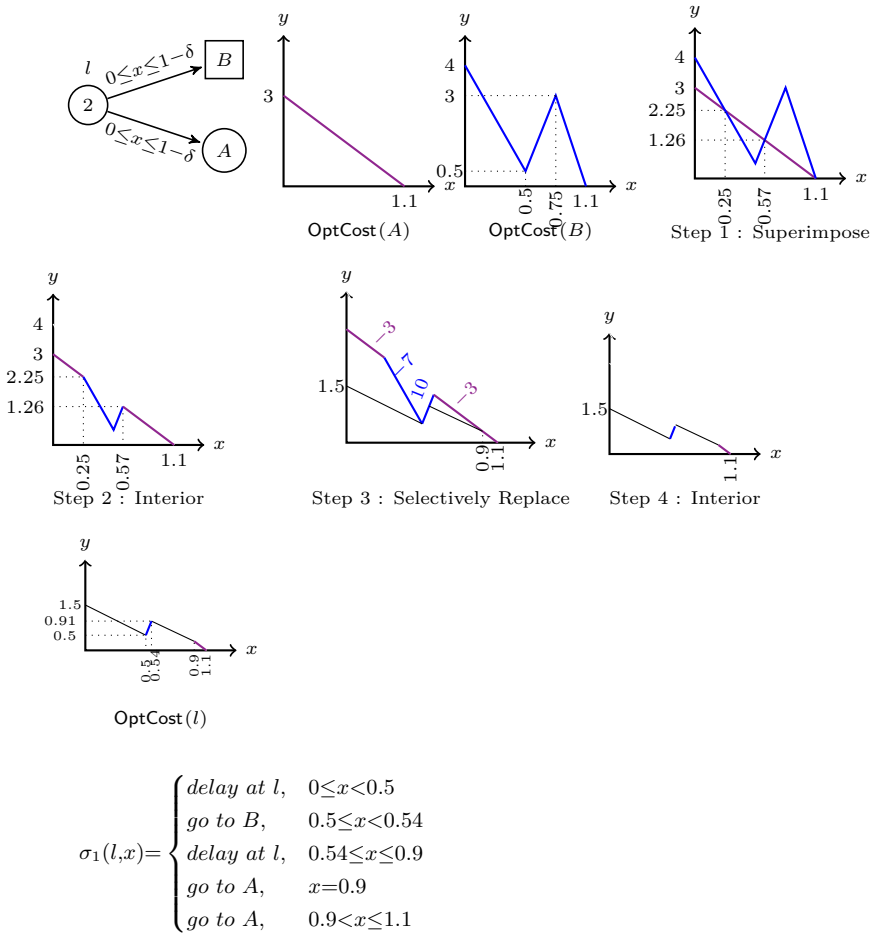
▶ **Lemma 5.5.** *In Algorithm 1, if a function g_i (in f of Step 2) has domain $[u_i, v_i]$ and slope $\leq -\eta(l)$ then $\mathsf{OptCost}(l, \nu) = (v_i - \nu) * \eta(l) + g(v_i)$.*

▶ **Lemma 5.6.** *The function f' in Algorithm 1 computes the optcost at any location l. That is, $\forall x \in [0, 1]$, $\mathsf{OptCost}_{\mathcal{G}}(l, x) = f'(x)$.*

Note that the strategy under construction is a player 1 strategy, and player 1 has no control over the interval $[1, 1+\delta]$. $x \in [1, 1+\delta]$ after a positive perturbation, and is under player 2's control. Thus, at a player 1 location, proving for $x \in [0, 1]$ suffices.

5.4.2 l_{min} is a Player 2 location

If l_{min} is a player 2 location in the reset-free component $\mathcal{G}_{\mathcal{F}_i}$, then intuitively, player 2 would want to spend as little time as possible there. Keeping this in mind, we first run



$$\sigma_1(l,x)=\begin{cases} \textit{delay at } l, & 0\leq x<0.5 \\ \textit{go to } B, & 0.5\leq x<0.54 \\ \textit{delay at } l, & 0.54\leq x\leq 0.9 \\ \textit{go to } A, & x=0.9 \\ \textit{go to } A, & 0.9<x\leq 1.1 \end{cases}$$

Figure 2 Optcost Computation for a Player 1 location ($\delta = 0.1$): we can keep the guards as $0 \leq x \leq 1$ and not apply Step 3 for $x \in [1-\delta, 1]$.

steps 1, 2 of Algorithm 1 by taking the exterior of $f_1, \ldots, f_n$ instead of the interior(player 2 would maximise the cost). There is no time elapse at l_{min} on running steps 1,2 of the algorithm. Let f be the computed exterior using steps 1,2. If f comprises of functions g_i having a greater slope than $-\eta(l)$, then Finally, while doing Step 4, we take the exterior of the replaced functions h_i and old functions g_i. Recall that our transformations resulted in 3 kinds of player 2 locations : urgent, those with dwell-time restriction $[0, \delta]$ and finally those with $[\delta, 2\delta]$. The 3 cases are discussed in detail in [10].

6 Conclusion and Future Work

In this paper we studied excess robust semantics and provided the first decidability result for excess semantics and improved the known undecidability result with 10 clocks to 5 clocks. To the best of our knowledge, the other known decidability result for robust timed games is under the conservative semantics for a fixed δ, [9]. As a consequence of our decidability result, the reachability problem for 1 clock PTG with arbitrary prices is shown to be decidable too under the assumption that the PTG does not have any negative cost cycle. The decidability we show

is for a fixed perturbation bound $\delta > 0$. We use δ in the constraints of the dwell-time PTG after the first transformation for ease of understanding the robust semantics. Implementing this in step 3 of Algorithm 1 and ensuring no time elapse in the interval $[1 - \delta, 1]$ takes no extra effort while l_{min} is a player 1 location. In that sense, we could have avoided explicit use of δ in the constraints in our simplifying transformations, and taken the appropriate steps in the algorithm itself. The existence of limit-strategy with $\delta \to 0$ seems rather hard. Our construction would not directly extend to limit-strategy problem as it is heavily dependant on the fixed δ.

References

1 R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In J. Díaz, J. Karhumäki, A. Lepistö, and D Sannella, editors, *Proc. ICALP'04*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

2 G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In M. D. Di Benedetto and A. L. Sangiovanni-Vincentelli, editors, *Proc. HSCC'01*, volume 2034 of *LNCS*, pages 147–161, Heidelberg, 2001. Springer.

3 P. Bouyer, T. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98:188–194, 2006.

4 P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In K. Lodaya and M. Mahajan, editors, *FSTTCS'04*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

5 Patricia Bouyer, Kim Guldstrand Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in one clock priced timed games. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, pages 345–356, 2006.

6 Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust reachability in timed automata: A game-based approach. In *Automata, Languages, and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2012.

7 Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust weighted timed automata and games. In Víctor Braberman and Laurent Fribourg, editors, *Proceedings of the 11th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'13)*, volume 8053 of *Lecture Notes in Computer Science*, pages 31–46, Buenos Aires, Argentina, August 2013. Springer.

8 T. Brihaye, V. Bruyère, and J. Raskin. On optimal timed strategies. In P. Pettersson and W. Yi, editors, *Proc. FORMATS'05*, volume 3829 of *LNCS*, pages 49–64. Springer, 2005.

9 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.

10 Shibashis Guha, Shankara Narayanan Krishna, Lakshmi Manasa, and Ashutosh Trivedi. Revisiting robustness in priced timed games. *CoRR*, abs/1507.05787, 2015.

11 T. D. Hansen, R. Ibsen-Jensen, and P. B. Miltersen. A faster algorithm for solving one-clock priced timed games. In PedroR. D'Argenio and Hernán Melgratti, editors, *CONCUR 2013 – Concurrency Theory*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.

12 N. Markey. Robustness in real-time systems. In *Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on*, pages 28–34, June 2011.

13 Michal Rutkowski. Two-player reachability-price games on single-clock timed automata. In *QAPL*, pages 31–46, 2011.

14 Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *CONCUR 2013 – Concurrency Theory*, volume 8052 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2013.

Simple Priced Timed Games are not That Simple*

Thomas Brihaye[1], Gilles Geeraerts[2], Axel Haddad[1], Engel Lefaucheux[3], and Benjamin Monmege[4]

1 **Université de Mons, Belgium,** {thomas.brihaye,axel.haddad}@umons.ac.be
2 **Université libre de Bruxelles, Belgium,** gigeerae@ulb.ac.be
3 **LSV, ENS Cachan, Inria Rennes, France,** engel.lefaucheux@ens-cachan.fr
4 **LIF, Aix-Marseille Université, CNRS, France,** benjamin.monmege@lif.univ-mrs.fr

Abstract

Priced timed games are two-player zero-sum games played on priced timed automata (whose locations and transitions are labeled by weights modeling the costs of spending time in a state and executing an action, respectively). The goals of the players are to minimise and maximise the cost to reach a target location, respectively. We consider priced timed games with one clock and arbitrary (positive and negative) weights and show that, for an important subclass of theirs (the so-called *simple* priced timed games), one can compute, in exponential time, the optimal values that the players can achieve, with their associated optimal strategies. As side results, we also show that one-clock priced timed games are determined and that we can use our result on simple priced timed games to solve the more general class of so-called reset-acyclic priced timed games (with arbitrary weights and one-clock).

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Priced timed games, Real-time systems, Game theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.278

1 Introduction

The importance of models inspired from the field of game theory is nowadays well-established in theoretical computer science. They allow to describe and analyse the possible interactions of antagonistic agents (or players) as in the *controller synthesis* problem, for instance. This problem asks, given a model of the environment of a system, and of the possible actions of a controller, to compute a controller that constraints the environment to respect a given specification. Clearly, one can not, in general, assume that the two players (the environment and the controller) will collaborate, hence the need to find a *controller strategy* that enforces the specification *whatever the environment does*. This question thus reduces to computing a so-called winning strategy for the corresponding player in the game model.

In order to describe precisely the features of complex computer systems, several game models have been considered in the literature. In this work, we focus on the model of Priced Timed Games [17] (PTGs for short), which can be regarded as an extension (in several directions) of classical finite automata. First, like timed automata [2], PTGs have *clocks*, which are real-valued variables whose values evolve with time elapsing, and which can be

* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under Grant Agreement n° 601148 (CASSTING).

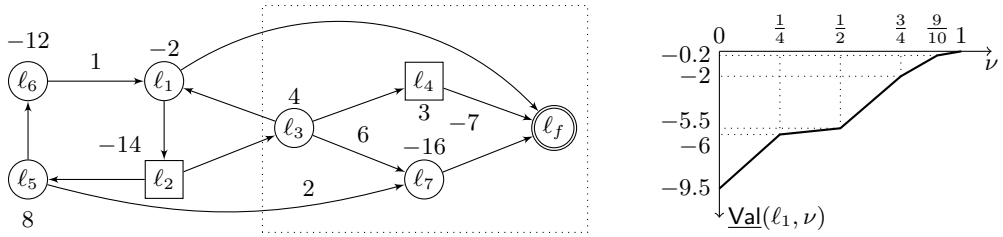


Figure 1 A simple priced timed game (left) and the lower value function of location ℓ_1 (right).

tested and reset along the transitions. Second, the locations are associated with price-rates and transitions are labeled by discrete prices, as in priced timed automata [4, 3, 6]. These prices allow one to associate a *cost* with each run (or play), which depends on the sequence of transitions traversed by the run, and on the time spent in each visited location. Finally, a PTG is played by two players, called Min and Max, and each location of the game is owned by either of them (we consider a turn-based version of the game). The player who controls the current location decides how long to wait, and which transition to take.

In this setting, the goal of Min is to reach a given set of target locations, following a play whose cost is as small as possible. Player Max has an antagonistic objective: he tries to avoid the target locations, and, if not possible, to maximise the accumulated cost up to the first visit of a target location. To reflect these objectives, we define the upper value $\overline{\mathsf{Val}}$ of the game as a mapping of the configurations of the PTG to the least cost that Min can guarantee while reaching the target, whatever the choices of Max. Similarly, the lower value $\underline{\mathsf{Val}}$ returns the greatest cost that Max can ensure (letting the cost being $+\infty$ in case the target locations are not reached).

An example of PTG is given in Figure 1, where the locations of Min (respectively, Max) are represented by circles (respectively, rectangles), and the integers next to the locations are their price-rates, i.e., the cost of spending one time unit in the location. Moreover, there is only one clock x in the game, which is never reset and all guards on transitions are $x \in [0,1]$ (hence this guard is not displayed and transitions are only labeled by their respective discrete cost): this is an example of *simple priced timed game* (we will define them properly later). It is easy to check that Min can force reaching the target location ℓ_f from all configurations (ℓ, ν) of the game, where ℓ is a location and ν is a real valuation of the clock in $[0,1]$. Let us comment on the optimal strategies for both players. From a configuration (ℓ_4, ν), with $\nu \in [0,1]$, Max better waits until the clock takes value 1, before taking the transition to ℓ_f (he is forced to move, by the rules of the game). Hence, Max's optimal value is $3(1-\nu) - 7 = -3\nu - 4$ from all configurations (ℓ_4, ν). Symmetrically, it is easy to check that Min better waits as long as possible in ℓ_7, hence his optimal value is $-16(1-\nu)$ from all configurations (ℓ_7, ν). However, optimal value functions are not always *that simple*, see for instance the lower value function of ℓ_1 on the right of Figure 1, which is a piecewise affine function. To understand why value functions can be piecewise affine, consider the sub-game enclosed in the dotted rectangle in Figure 1, and consider the value that Min can guarantee from a configuration of the form (ℓ_3, ν) in this sub-game. Clearly, Min must decide how long he will spend in ℓ_3 and whether he will go to ℓ_4 or ℓ_7. His optimal value from all (ℓ_3, ν) is thus $\inf_{0 \leqslant t \leqslant 1-\nu} \min\big(4t + (-3(\nu+t) - 4), 4t + 6 - 16(1-(\nu+t))\big) = \min(-3\nu - 4, 16\nu - 10)$. Since $16\nu - 10 \geqslant -3\nu - 4$ if and only if $\nu \leqslant 6/19$, the best choice of Min is to move instantaneously to ℓ_7 if $\nu \in [0, 6/19]$ and to move instantaneously to ℓ_4 if $\nu \in (6/19, 1]$, hence the value function of ℓ_3 (in the subgame) is a piecewise affine function with two pieces.

Related work. PTGs were independently investigated in [8] and [1]. For (non-necessarily turn-based) PTGs with *non-negative* prices, semi-algorithms are given to decide the *value problem* that is to say, whether the upper value of a location (the best cost that Min can guarantee in valuation 0), is below a given threshold. They have also shown that, under the *strongly non-Zeno assumption* on prices (asking the existence of $\kappa > 0$ such that every cycle in the underlying region graph has a cost at least κ), the proposed semi-algorithms always terminate. This assumption was justified in [11, 7] by showing that, without it, the *existence problem*, that is to decide whether Min has a strategy guaranteeing to reach a target location with a cost below a given threshold, is indeed undecidable for PTGs with non-negative prices and three or more clocks. This result was recently extended in [9] to show that the *value problem* is also undecidable for PTGs with non-negative prices and four or more clocks. In [5], the undecidability of the existence problem has also been shown for PTGs with arbitrary price-rates (without prices on transitions), and two or more clocks. On a positive side, the value problem was shown decidable by [10] for PTGs with one clock when the prices are non-negative: a 3-exponential time algorithm was first proposed, further refined in [18, 16] into an exponential time algorithm. The key point of those algorithms is to reduce the problem to the computation of optimal values in a restricted family of PTGs called *Simple Priced Timed Games* (SPTGs for short), where the underlying automata contain no guard, no reset, and the play is forced to stop after one time unit. More precisely, the PTG is decomposed into a sequence of SPTGs whose value functions are computed and re-assembled to yield the value function of the original PTG. Alternatively, and with radically different techniques, a pseudo-polynomial time algorithm to solve one-clock PTGs with arbitrary prices on transitions, and price-rates restricted to two values amongst $\{-d, 0, +d\}$ (with $d \in \mathbf{N}$) was given in [14].

Contributions. Following the decidability results sketched above, we consider PTGs with one clock. We extend those results by considering arbitrary (positive and negative) prices. Indeed, all previous works on PTGs with only one clock (except [14]) have considered non-negative weights only, and the status of the more general case with arbitrary weights has so far remained elusive. Yet, arbitrary weights are an important modeling feature. Consider, for instance, a system which can consume but also produce energy at different rates. In this case, energy consumption could be modeled as a positive price-rate, and production by a negative price-rate. We propose an *exponential time algorithm to compute the value of one-clock* SPTG*s with arbitrary weights*. While this result might sound limited due to the restricted class of simple PTGs we can handle, we recall that the previous works mentioned above [10, 18, 16] have demonstrated that solving SPTGs is a key result towards solving more general PTGs. Moreover, this algorithm is, as far as we know, the first to handle the full class of SPTGs with arbitrary weights, and we note that the solutions (either the algorithms or the proofs) known so far do not generalise to this case. Finally, as a side result, this algorithm allows us to solve the more general class of *reset-acyclic* one-clock PTGs that we introduce. Thus, although we can not (yet) solve the whole class of PTGs with arbitrary weights, our result may be seen as a potentially important milestone towards this goal.

Due to lack of space most proofs and technical details may be found in a detailed version [12].

2 Priced timed games: syntax, semantics, and preliminary results

Notations and definitions. Let x denote a positive real-valued variable called *clock*. A *guard* (or *clock constraint*) is an interval with endpoints in $\mathbf{N} \cup \{+\infty\}$. We often abbreviate

guards, for instance $x \leqslant 5$ instead of $[0,5]$. Let $S \subseteq \mathsf{Guard}(x)$ be a finite set of guards. We let $[\![S]\!] = \bigcup_{I \in S} I$. Assuming $M_0 = 0 < M_1 < \cdots < M_k$ are all the endpoints of the intervals in S (to which we add 0), we let $\mathsf{Reg}_S = \{(M_i, M_{i+1}) \mid 0 \leqslant i \leqslant k-1\} \cup \{\{M_i\} \mid 0 \leqslant i \leqslant k\}$ be the set of *regions* of S. Observe that Reg_S is also a set of guards.

We rely on the notion of *cost function* to formalise the notion of optimal value function sketched in the introduction. Formally, for a set of guards $S \subseteq \mathsf{Guard}(x)$, a *cost function* over S is a function $f\colon [\![\mathsf{Reg}_S]\!] \to \overline{\mathbf{R}} = \mathbf{R} \cup \{+\infty, -\infty\}$ such that over all regions $r \in \mathsf{Reg}_S$, f is either infinite or a continuous piecewise affine function, with a finite set of cutpoints (points where the first derivative is not defined) $\{\kappa_1, \ldots, \kappa_p\} \subseteq \mathbf{Q}$, and with $f(\kappa_i) \in \mathbf{Q}$ for all $1 \leqslant i \leqslant p$. In particular, if $f(r) = \{f(\nu) \mid \nu \in r\}$ contains $+\infty$ (respectively, $-\infty$) for some region r, then $f(r) = \{+\infty\}$ ($f(r) = \{-\infty\}$). We denote by CF_S the set of all cost functions over S. In our algorithm to solve SPTGs, we will need to combine cost functions thanks to the $\rhd$ operator. Let $f \in \mathsf{CF}_S$ and $f' \in \mathsf{CF}_{S'}$ be two costs functions on set of guards $S, S' \subseteq \mathsf{Guard}(x)$, such that $[\![S]\!] \cap [\![S']\!]$ is a singleton. We let $f \rhd f'$ be the cost function in $\mathsf{CF}_{S \cup S'}$ such that $(f \rhd f')(\nu) = f(\nu)$ for all $\nu \in [\![\mathsf{Reg}_S]\!]$, and $(f \rhd f')(\nu) = f'(\nu)$ for all $\nu \in [\![\mathsf{Reg}_{S'}]\!] \setminus [\![\mathsf{Reg}_S]\!]$.

We consider an extended notion of one-clock priced timed games (PTGs for short) allowing for the use of *urgent locations*, where only a zero delay can be spent, and *final cost functions* which are associated with each final location and incur an extra cost to be paid when ending the game in this location. Formally, a PTG $\mathcal{G}$ is a tuple $(L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$ where

L_{Min} (respectively, L_{Max}) is a finite set of *locations* for player Min (respectively, Max), with $L_{\mathsf{Min}} \cap L_{\mathsf{Max}} = \emptyset$; L_f is a finite set of *final* locations, and we let $L = L_{\mathsf{Min}} \cup L_{\mathsf{Max}} \cup L_f$ be the whole location space; $L_u \subseteq L \setminus L_f$ indicates *urgent* locations[1];

$\Delta \subseteq (L \setminus L_f) \times \mathsf{Guard}(x) \times \{\top, \bot\} \times L$ is a finite set of *transitions*; $\boldsymbol{\varphi} = (\varphi_\ell)_{\ell \in L_f}$ associates to each $\ell \in L_f$ its *final cost function*, that is an affine[2] cost function φ_ℓ over $S_{\mathcal{G}} = \{I \mid \exists \ell, R, \ell' : (\ell, I, R, \ell') \in \Delta\}$; $\pi\colon L \cup \Delta \to \mathbf{Z}$ mapping an integer *price* to each location – its *price-rate* – and transition.

Intuitively, a transition (ℓ, I, R, ℓ') changes the current location from ℓ to ℓ' if the clock has value in I and the clock is reset according to the Boolean R. We assume that, in all PTGs, the clock x is *bounded*, i.e., there is $M \in \mathbf{N}$ such that for all guards $I \in S_{\mathcal{G}}$, $I \subseteq [0, M]$.[3] We denote by $\mathsf{Reg}_{\mathcal{G}}$ the set $\mathsf{Reg}_{S_{\mathcal{G}}}$ of *regions of* $\mathcal{G}$. We further denote[4] by $\Pi^{\mathrm{tr}}_{\mathcal{G}}$, $\Pi^{\mathrm{loc}}_{\mathcal{G}}$ and $\Pi^{\mathrm{fin}}_{\mathcal{G}}$ respectively the values $\max_{\delta \in \Delta} |\pi(\delta)|$, $\max_{\ell \in L} |\pi(\ell)|$ and $\sup_{\nu \in [0,M]} \max_{\ell \in L} |\varphi_\ell(\nu)| = \max_{\ell \in L} \max(|\varphi_\ell(0)|, |\varphi_\ell(M)|)$. That is, $\Pi^{\mathrm{tr}}_{\mathcal{G}}$, $\Pi^{\mathrm{loc}}_{\mathcal{G}}$ and $\Pi^{\mathrm{fin}}_{\mathcal{G}}$ are the largest absolute values of the location prices, transition prices and final cost functions.

Let $\mathcal{G} = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$ be a PTG. A *configuration* of $\mathcal{G}$ is a pair $s = (\ell, \nu) \in L \times \mathbf{R}^+$. We denote by $\mathsf{Conf}_{\mathcal{G}}$ the set of configurations of $\mathcal{G}$. Let (ℓ, ν) and (ℓ', ν') be two configurations. Let $\delta = (\ell, I, R, \ell') \in \Delta$ be a transition of $\mathcal{G}$ and $t \in \mathbf{R}^+$ be a delay. Then, there is a (t, δ)-transition from (ℓ, ν) to (ℓ', ν') with cost c, denoted by $(\ell, \nu) \xrightarrow{t, \delta, c} (\ell', \nu')$, if

[1] Here we differ from [10] where $L_u \subseteq L_{\mathsf{Max}}$.

[2] The affine restriction on final cost function is to simplify our further arguments, though we do believe that all of our results could be adapted to cope with general cost functions.

[3] Observe that this last restriction is *not* without loss of generality in the case of PTGs. While all timed automata $\mathcal{A}$ can be turned into an equivalent (with respect to reachability properties) $\mathcal{A}'$ whose clocks are bounded [4], this technique can not be applied to PTGs, in particular with arbitrary prices.

[4] Throughout the paper, we often drop the $\mathcal{G}$ in the subscript of several notations when the game is clear from the context.

(i) $\ell \in L_u$ implies $t = 0$;
(ii) $\nu + t \in I$;
(iii) $R = \top$ implies $\nu' = 0$;
(iv) $R = \bot$ implies $\nu' = \nu + t$;
(v) $c = \pi(\delta) + t \times \pi(\ell)$.

Observe that the cost of (t, δ) takes into account the price-rate of ℓ, the delay spent in ℓ, and the price of δ. We assume that the game has no deadlock: for all $s \in \mathsf{Conf}_{\mathcal{G}}$, there are (t, δ, c) and $s' \in \mathsf{Conf}_{\mathcal{G}}$ such that $s \xrightarrow{t,\delta,c} s'$. Finally, we write $s \xrightarrow{c} s'$ whenever there are t and δ such that $s \xrightarrow{t,\delta,c} s'$. A *play* of $\mathcal{G}$ is a finite or infinite path $\rho = (\ell_0, \nu_0) \xrightarrow{c_0} (\ell_1, \nu_1) \xrightarrow{c_1} (\ell_2, \nu_2) \cdots$. For a finite play $\rho = (\ell_0, \nu_0) \xrightarrow{c_0} (\ell_1, \nu_1) \xrightarrow{c_1} (\ell_2, \nu_2) \cdots \xrightarrow{c_{n-1}} (\ell_n, \nu_n)$, we let $|\rho| = n$. For an infinite play $\rho = (\ell_0, \nu_0) \xrightarrow{c_0} (\ell_1, \nu_1) \xrightarrow{c_1} (\ell_2, \nu_2) \cdots$, we let $|\rho|$ be the least position i such that $\ell_i \in L_f$ if such a position exists, and $|\rho| = +\infty$ otherwise. Then, we let $\mathsf{Cost}_{\mathcal{G}}(\rho)$ be the *cost* of ρ, with $\mathsf{Cost}_{\mathcal{G}}(\rho) = +\infty$ if $|\rho| = +\infty$, and $\mathsf{Cost}_{\mathcal{G}}(\rho) = \sum_{i=0}^{|\rho|-1} c_i + \varphi_{\ell_{|\rho|}}(\nu_{|\rho|})$ otherwise.

A *strategy* for player Min is a function σ_{Min} mapping every finite play ending in location of Min to a pair $(t, \delta) \in \mathbf{R}^+ \times \Delta$, indicating what Min should play. We also request that the strategy proposes only valid pairs (t, δ), i.e., that for all runs ρ ending in (ℓ, ν), $\sigma_{\mathsf{Min}}(\rho) = (t, (\ell, I, R, \ell'))$ implies that $\nu + t \in I$. Strategies σ_{Max} of player Max are defined accordingly. We let $\mathsf{Strat}_{\mathsf{Min}}(\mathcal{G})$ and $\mathsf{Strat}_{\mathsf{Max}}(\mathcal{G})$ be the sets of strategies of Min and Max, respectively. A pair of strategies $(\sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}}) \in \mathsf{Strat}_{\mathsf{Min}}(\mathcal{G}) \times \mathsf{Strat}_{\mathsf{Max}}(\mathcal{G})$ is called a *profile of strategies*. Together with an initial configuration $s_0 = (\ell_0, \nu_0)$, it defines a unique play $\mathsf{Play}(s_0, \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}}) = s_0 \xrightarrow{c_0} s_1 \xrightarrow{c_1} s_2 \cdots s_k \xrightarrow{c_k} \cdots$ where for all $j \geqslant 0$, s_{j+1} is the unique configuration such that $s_j \xrightarrow{t_j,\delta_j,c_j} s_{j+1}$ with $(t_j, \delta_j) = \sigma_{\mathsf{Min}}(s_0 \xrightarrow{c_0} s_1 \cdots s_{j-1} \xrightarrow{c_{j-1}} s_j)$ if $\ell_j \in L_{\mathsf{Min}}$; and $(t_j, \delta_j) = \sigma_{\mathsf{Max}}(s_0 \xrightarrow{c_0} s_1 \cdots s_{j-1} \xrightarrow{c_{j-1}} s_j)$ if $\ell_j \in L_{\mathsf{Max}}$. We let $\mathsf{Play}(\sigma_{\mathsf{Min}})$ (respectively, $\mathsf{Play}(s_0, \sigma_{\mathsf{Min}})$) be the set of plays that conform with σ_{Min} (and start in s_0).

As sketched in the introduction, we consider optimal reachability-cost games on PTGs, where the aim of player Min is to reach a location of L_f while minimising the cost. To formalise this objective, we let the value of a strategy σ_{Min} for Min be the function $\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}} \colon \mathsf{Conf}_{\mathcal{G}} \to \overline{\mathbf{R}}$ such that for all $s \in \mathsf{Conf}_{\mathcal{G}}$: $\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}}(s) = \sup_{\sigma_{\mathsf{Max}} \in \mathsf{Strat}_{\mathsf{Max}}} \mathsf{Cost}(\mathsf{Play}(s, \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}}))$. Intuitively, $\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}}(s)$ is the largest value that Max can achieve when playing against strategy σ_{Min} of Min (it is thus a worst case from the point of view of Min). Symmetrically, for $\sigma_{\mathsf{Max}} \in \mathsf{Strat}_{\mathsf{Max}}$, $\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Max}}}(s) = \inf_{\sigma_{\mathsf{Min}} \in \mathsf{Strat}_{\mathsf{Min}}} \mathsf{Cost}(\mathsf{Play}(s, \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}}))$, for all $s \in \mathsf{Conf}_{\mathcal{G}}$. Then, the *upper and lower values* of $\mathcal{G}$ are respectively the functions $\overline{\mathsf{Val}}_{\mathcal{G}} \colon \mathsf{Conf}_{\mathcal{G}} \to \overline{\mathbf{R}}$ and $\underline{\mathsf{Val}}_{\mathcal{G}} \colon \mathsf{Conf}_{\mathcal{G}} \to \overline{\mathbf{R}}$ where, for all $s \in \mathsf{Conf}_{\mathcal{G}}$, $\overline{\mathsf{Val}}_{\mathcal{G}}(s) = \inf_{\sigma_{\mathsf{Min}} \in \mathsf{Strat}_{\mathsf{Min}}} \mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}}(s)$ and $\underline{\mathsf{Val}}_{\mathcal{G}}(s) = \sup_{\sigma_{\mathsf{Max}} \in \mathsf{Strat}_{\mathsf{Max}}} \mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Max}}}(s)$. We say that a game is *determined* if the lower and the upper values match for every configuration s, and in this case, we say that the optimal value $\mathsf{Val}_{\mathcal{G}}$ of the game $\mathcal{G}$ exists, defined by $\mathsf{Val}_{\mathcal{G}} = \underline{\mathsf{Val}}_{\mathcal{G}} = \overline{\mathsf{Val}}_{\mathcal{G}}$. A strategy σ_{Min} of Min is *optimal* (respectively, *ε-optimal*) if $\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}} = \overline{\mathsf{Val}}_{\mathcal{G}}$ ($\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}} \leqslant \overline{\mathsf{Val}}_{\mathcal{G}} + \varepsilon$), i.e., σ_{Min} ensures that the cost of the plays will be at most $\overline{\mathsf{Val}}_{\mathcal{G}}$ ($\overline{\mathsf{Val}}_{\mathcal{G}} + \varepsilon$). Symmetrically, a strategy σ_{Max} of Max is *optimal* (respectively, *ε-optimal*) if $\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Max}}} = \underline{\mathsf{Val}}_{\mathcal{G}}$ ($\mathsf{Val}_{\mathcal{G}}^{\sigma_{\mathsf{Max}}} \geqslant \underline{\mathsf{Val}}_{\mathcal{G}} - \varepsilon$).

Properties of the value. Let us now prove useful preliminary properties of the value function of PTGs, that – as far as we know – had hitherto never been established. Using a general determinacy result by Gale and Stewart [15], we can show that PTGs (with one clock) are *determined*. Hence, the value function $\mathsf{Val}_{\mathcal{G}}$ exists for all PTG $\mathcal{G}$. We can further show that, for all locations ℓ, $\mathsf{Val}_{\mathcal{G}}(\ell)$ is a *piecewise continuous function* that might exhibit discontinuities *only on the borders of the regions* of $\mathsf{Reg}_{\mathcal{G}}$ (where $\mathsf{Val}_{\mathcal{G}}(\ell)$ is the function such that $\mathsf{Val}_{\mathcal{G}}(\ell)(\nu) = \mathsf{Val}_{\mathcal{G}}(\ell, \nu)$ for all $\nu \in \mathbf{R}^+$).

▶ **Theorem 1.** *For all (one-clock)* PTG*s* $\mathcal{G}$*:*
(i) $\overline{\mathsf{Val}}_{\mathcal{G}} = \underline{\mathsf{Val}}_{\mathcal{G}}$*, i.e.,* PTG*s are* determined*; and*
(ii) *for all* $r \in \mathsf{Reg}_{\mathcal{G}}$*, for all* $\ell \in L$*,* $\mathsf{Val}_{\mathcal{G}}(\ell)$ *is either infinite or continuous over* r*.*

Simple priced timed games. As sketched in the introduction, our main contribution is to solve the special case of simple one-clock priced timed games with arbitrary costs. Formally, an r-SPTG, with $r \in \mathbf{Q}^+ \cap [0,1]$, is a PTG $\mathcal{G} = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$ such that for all transitions $(\ell, I, R, \ell') \in \Delta$, $I = [0, r]$ and $R = \bot$. Hence, transitions of r-SPTGs are henceforth denoted by (ℓ, ℓ'), dropping the guard and the reset. Then, an SPTG is a 1-SPTG. This paper is devoted mainly to proving the following theorem on SPTGs:

▶ **Theorem 2.** *Let* $\mathcal{G}$ *be an* SPTG*. Then, for all locations* $\ell \in L$*, the function* $\mathsf{Val}_{\mathcal{G}}(\ell)$ *is either infinite, or continuous and piecewise-affine with at most an exponential number of cutpoints. The value functions for all locations, as well as a pair of optimal strategies* $(\sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$ *(that always exist if no values are infinite) can be computed in exponential time.*

Before sketching the proof of this theorem, we discuss a class of (simple) strategies that are sufficient to play optimally. Roughly speaking, Max has always a *memoryless* optimal strategy, while Min might need *(finite) memory* to play optimally – it is already the case in untimed quantitative reachability games with arbitrary weights [13]. Moreover, these strategies are finitely representable (recall that even a memoryless strategy depends on the current *configuration* and that there are infinitely many in our time setting).

Strategies of Max are formalised with the notion of *finite positional strategies* (FP-strategies): they are memoryless strategies σ (i.e., for all finite plays $\rho_1 = \rho_1' \xrightarrow{c_1} s$ and $\rho_2 = \rho_2' \xrightarrow{c_2} s$ ending in the same configuration, we have $\sigma(\rho_1) = \sigma(\rho_2)$), such that for all locations ℓ, there exists a finite sequence of rationals $0 \leqslant \nu_1^\ell < \nu_2^\ell < \cdots < \nu_k^\ell = 1$ and a finite sequence of transitions $\delta_1, \ldots, \delta_k \in \Delta$ such that
(i) for all $1 \leqslant i \leqslant k$, for all $\nu \in (\nu_{i-1}^\ell, \nu_i^\ell]$, either $\sigma(\ell, \nu) = (0, \delta_i)$, or $\sigma(q, \nu) = (\nu_i^\ell - \nu, \delta_i)$ (assuming $\nu_0^\ell = \min(0, \nu_1^\ell)$); and
(ii) if $\nu_1^\ell > 0$, then $\sigma(\ell, 0) = (\nu_1^\ell, \delta_1)$.
We let $\mathsf{pts}(\sigma)$ be the set of ν_i^ℓ for all ℓ and i, and $\mathsf{int}(\sigma)$ be the set of all successive intervals generated by $\mathsf{pts}(\sigma)$. Finally, we let $|\sigma| = |\mathsf{int}(\sigma)|$ be the size of σ. Intuitively, in an interval $(\nu_{i-1}^\ell, \nu_i^\ell]$, σ always returns the same move: either to take *immediately* δ_i or to wait until the clock reaches the endpoint ν_i^ℓ and then take δ_i.

Min, however may require memory to play optimally. Informally, we will compute optimal *switching* strategies, as introduced in [13] (in the untimed setting). A switching strategy is described by a pair $(\sigma_{\mathsf{Min}}^1, \sigma_{\mathsf{Min}}^2)$ of FP-strategies and a switch threshold K, and consists in playing σ_{Min}^1 until the total accumulated cost of the discrete transitions is below K; and then to *switch* to strategy σ_{Min}^2. The role of σ_{Min}^2 is to ensure reaching a final location: it is thus a (classical) attractor strategy. The role of σ_{Min}^1, on the other hand, is to allow Min to decrease the cost low enough (possibly by forcing negative cycles) to secure a cost below K, and the computation of σ_{Min}^1 is thus the critical point in the computation of an optimal switching strategy. To characterise σ_{Min}^1, we introduce the notion of negative cycle strategy (NC-strategy). Formally, an NC-strategy σ_{Min} of Min is an FP-strategy such that for all runs $\rho = (\ell_1, \nu) \xrightarrow{c_1} \cdots \xrightarrow{c_{k-1}} (\ell_k, \nu') \in \mathsf{Play}(\sigma_{\mathsf{Min}})$ with $\ell_1 = \ell_k$, and ν, ν' in the same interval of $\mathsf{int}(\sigma_{\mathsf{Min}})$, the sum of prices of *discrete transitions* is at most -1, i.e., $\pi(\ell_1, \ell_2) + \cdots + \pi(\ell_{k-1}, \ell_k) \leqslant -1$. To characterise the fact that σ_{Min}^1 must allow Min to reach a cost which is *small enough, without necessarily reaching a target state*, we define the *fake value* of an NC-strategy σ_{Min} from a configuration s as $\mathsf{fake}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}}(s) = \sup\{\mathsf{Cost}(\rho) \mid$

$\rho \in \mathsf{Play}(s, \sigma_{\mathsf{Min}}), \rho$ reaches a target$\}$, i.e., the value obtained when *ignoring* the σ_{Min}-induced plays that *do not* reach the target. Thus, clearly, $\mathsf{fake}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}}(s) \leqslant \mathsf{Val}^{\sigma_{\mathsf{Min}}}(s)$. We say that an NC-strategy is *fake-optimal* if its fake value, in every configuration, is equal to the optimal value of the configuration in the game. This is justified by the following result whose proof relies on the switching strategies described before:

▶ **Lemma 3.** *If* $\mathsf{Val}_{\mathcal{G}}(\ell, \nu) \neq +\infty$*, for all* ℓ *and* ν*, then for all NC-strategies* σ_{Min}*, there is a strategy* σ'_{Min} *such that* $\mathsf{Val}_{\mathcal{G}}^{\sigma'_{\mathsf{Min}}}(s) \leqslant \mathsf{fake}_{\mathcal{G}}^{\sigma_{\mathsf{Min}}}(s)$ *for all configurations* s*. In particular, if* σ_{Min} *is a fake-optimal NC-strategy, then* σ'_{Min} *is an optimal (switching) strategy of the* SPTG*.*

Then, an SPTG is called *finitely optimal* if

(i) Min has a fake-optimal NC-strategy;
(ii) Max has an optimal FP-strategy; and
(iii) $\mathsf{Val}_{\mathcal{G}}(\ell)$ is a cost function, for all locations ℓ.

The central point in establishing Theorem 2 will thus be to prove that **all SPTGs are finitely optimal**, as this guarantees the existence of well-behaved optimal strategies and value functions. We will also show that they can be computed in exponential time. The proof is by induction on the number of urgent locations of the SPTG. In Section 3, we address the base case of SPTGs with urgent locations only (where no time can elapse). Since these SPTGs are very close to the *untimed* min-cost reachability games of [13], we adapt the algorithm in this work and obtain the `solveInstant` function (Algorithm 1). This function can also compute $\mathsf{Val}_{\mathcal{G}}(\ell, 1)$ for all ℓ and all games $\mathcal{G}$ (even with non-urgent locations) since time can not elapse anymore when the clock has valuation 1. Next, using the continuity result of Theorem 1, we can detect locations ℓ where $\mathsf{Val}_{\mathcal{G}}(\ell, \nu) \in \{+\infty, -\infty\}$, for all $\nu \in [0, 1]$, and remove them from the game. Finally, in Section 4 we handle SPTGs with non-urgent locations by refining the technique of [10, 18] (that work only on SPTGs with non-negative costs). Compared to [10, 18], our algorithm is simpler, being iterative, instead of recursive.

3 SPTGs with only urgent locations

Throughout this section, we consider an r-SPTG $\mathcal{G} = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$ where all locations are urgent, i.e., $L_u = L_{\mathsf{Min}} \cup L_{\mathsf{Max}}$. We first explain briefly how we can compute the value function of the game for a *fixed* clock valuation $\nu \in [0, r]$ (more precisely, we can compute the vector $(\mathsf{Val}_{\mathcal{G}}(\ell, \nu))_{\ell \in L}$). Since no time can elapse, we can adapt the techniques developed in [13] to solve (untimed) *min-cost reachability games*. The adaptation consists in taking into account the final cost functions. This yields the function `solveInstant` (Algorithm 1), that computes the vector $(\mathsf{Val}_{\mathcal{G}}(\ell, \nu))_{\ell \in L}$ for a fixed ν. The results of [13] also allow us to compute associated optimal strategies: when $\mathsf{Val}(\ell, \nu) \notin \{-\infty, +\infty\}$ the optimal strategy for Max is memoryless, and the optimal strategy for Min is a switching strategy $(\sigma^1_{\mathsf{Min}}, \sigma^2_{\mathsf{Min}})$ with a threshold K (as described in the previous section).

Now let us explain how we can reduce the computation of $\mathsf{Val}_{\mathcal{G}}(\ell) \colon \nu \in [0, r] \mapsto \mathsf{Val}(\ell, \nu)$ (for all ℓ) to a *finite number of calls* to `solveInstant`. Let $\mathsf{F}_{\mathcal{G}}$ be the set of affine functions over $[0, r]$ such that $\mathsf{F}_{\mathcal{G}} = \{k + \varphi_\ell \mid \ell \in L_f \wedge k \in \mathcal{I}\}$, where $\mathcal{I} = [-(|L| - 1)\Pi^{\mathrm{tr}}, |L|\Pi^{\mathrm{tr}}] \cap \mathbf{Z}$. Observe that $\mathsf{F}_{\mathcal{G}}$ has cardinality $2|L|^2\Pi^{\mathrm{tr}}$, i.e., pseudo-polynomial in the size of $\mathcal{G}$. From [13], we conclude that the functions in $\mathsf{F}_{\mathcal{G}}$ are sufficient to characterise $\mathsf{Val}_{\mathcal{G}}$, in the following sense: for all $\ell \in L$ and $\nu \in [0, r]$ such that $\mathsf{Val}(\ell, \nu) \notin \{-\infty, +\infty\}$, there is $f \in \mathsf{F}_{\mathcal{G}}$ with $\mathsf{Val}(\ell, \nu) = f(\nu)$. Using the continuity of $\mathsf{Val}_{\mathcal{G}}$ (Theorem 1), we show that all the cutpoints of $\mathsf{Val}_{\mathcal{G}}$ are intersections of functions from $\mathsf{F}_{\mathcal{G}}$, i.e., belong to the set of *possible cutpoints* $\mathsf{PossCP}_{\mathcal{G}} = \{\nu \in [0, r] \mid \exists f_1, f_2 \in \mathsf{F}_{\mathcal{G}} \quad f_1 \neq f_2 \wedge f_1(\nu) = f_2(\nu)\}$. Observe that $\mathsf{PossCP}_{\mathcal{G}}$

Algorithm 1: `solveInstant`$(\mathcal{G},\nu)$

Input: r-SPTG $\mathcal{G} = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$, a valuation $\nu \in [0, r]$

foreach $\ell \in L$ **do**
  if $\ell \in L_f$ **then** $\mathsf{X}(\ell) := \varphi_\ell(\nu)$ **else** $\mathsf{X}(\ell) := +\infty$
repeat
  $\mathsf{X}_{pre} := \mathsf{X}$
  foreach $\ell \in L_{\mathsf{Max}}$ **do** $\mathsf{X}(\ell) := \max_{(\ell,\ell')\in\Delta} \left(\pi(\ell,\ell') + \mathsf{X}_{pre}(\ell')\right)$
  foreach $\ell \in L_{\mathsf{Min}}$ **do** $\mathsf{X}(\ell) := \min_{(\ell,\ell')\in\Delta} \left(\pi(\ell,\ell') + \mathsf{X}_{pre}(\ell')\right)$
  foreach $\ell \in L$ such that $\mathsf{X}(\ell) < -(|L|-1)\Pi^{\mathrm{tr}} - \Pi^{\mathrm{fin}}$ **do** $\mathsf{X}(\ell) := -\infty$
until $\mathsf{X} = \mathsf{X}_{pre}$
return X

contains at most $|\mathsf{F}_{\mathcal{G}}|^2 = 4|L_f|^4(\Pi^{\mathrm{tr}})^2$ points (also a pseudo-polynomial in the size of $\mathcal{G}$) since all functions in $\mathsf{F}_{\mathcal{G}}$ are affine, and can thus intersect at most once with every other function. Moreover, $\mathsf{PossCP}_{\mathcal{G}} \subseteq \mathbf{Q}$, since all functions of $\mathsf{F}_{\mathcal{G}}$ take rational values in 0 and $r \in \mathbf{Q}$. Thus, for all ℓ, $\mathsf{Val}_{\mathcal{G}}(\ell)$ is a cost function (with cutpoints in $\mathsf{PossCP}_{\mathcal{G}}$ and pieces from $\mathsf{F}_{\mathcal{G}}$). Since $\mathsf{Val}_{\mathcal{G}}(\ell)$ is a piecewise affine function, we can characterise it completely by computing only its value on its cutpoints. Hence, we can reconstruct $\mathsf{Val}_{\mathcal{G}}(\ell)$ by calling `solveInstant` on each rational valuation $\nu \in \mathsf{PossCP}_{\mathcal{G}}$. From the optimal strategies computed along `solveInstant` [13], we can also reconstruct a fake-optimal NC-strategy for Min and an optimal FP-strategy for Max, hence:

▶ **Proposition 4.** *Every r-SPTG $\mathcal{G}$ with only urgent locations is finitely optimal. Moreover, for all locations ℓ, the piecewise affine function $\mathsf{Val}_{\mathcal{G}}(\ell)$ has cutpoints in $\mathsf{PossCP}_{\mathcal{G}}$ of cardinality $4|L_f|^4(\Pi^{\mathrm{tr}})^2$, pseudo-polynomial in the size of $\mathcal{G}$.*

4 Solving general SPTGs

In this section, we consider SPTGs with possibly non-urgent locations. We first prove that all such SPTGs are finitely optimal. Then, we introduce Algorithm 2 to compute optimal values and strategies of SPTGs. To the best of our knowledge, this is the first algorithm to solve SPTGs with arbitrary weights. Throughout the section, we fix an SPTG $\mathcal{G} = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$ with possibly non-urgent locations. Before presenting our core contributions, let us explain how we can detect locations with infinite values. As already argued, we can compute $\mathsf{Val}(\ell, 1)$ for all ℓ assuming all locations are urgent, since time can not elapse anymore when the clock has valuation 1. This can be done with `solveInstant`. Then, by continuity, $\mathsf{Val}(\ell, 1) = +\infty$ (respectively, $\mathsf{Val}(\ell, 1) = -\infty$), for some ℓ if and only if $\mathsf{Val}(\ell, \nu) = +\infty$ (respectively, $\mathsf{Val}(\ell, \nu) = -\infty$) for all $\nu \in [0, 1]$. We remove from the game all locations with infinite value without changing the values of other locations (as justified in [13]). Thus, we henceforth assume that $\mathsf{Val}(\ell, \nu) \in \mathbf{R}$ for all (ℓ, ν).

The $\mathcal{G}_{L',r}$ construction. To prove finite optimality of SPTGs and to establish correctness of our algorithm, we rely in both cases on a construction that consists in decomposing $\mathcal{G}$ into a sequence of SPTGs with *more urgent locations*. Intuitively, a game with more urgent locations is easier to solve since it is closer to an untimed game (in particular, when all locations are urgent, we can apply the techniques of Section 3). More precisely, given a set L' of non-urgent locations, and a valuation $r_0 \in [0, 1]$, we will define a (possibly infinite)

sequence of valuations $1 = r_0 > r_1 > \cdots$ and a sequence $\mathcal{G}_{L',r_0}$, $\mathcal{G}_{L',r_1}, \ldots$ of SPTGs such that

- **(i)** all locations of $\mathcal{G}$ are also present in each $\mathcal{G}_{L',r_i}$, except that the locations of L' are now urgent; and
- **(ii)** for all $i \geqslant 0$, the value function of $\mathcal{G}_{L',r_i}$ is equal to $\mathsf{Val}_{\mathcal{G}}$ on the interval $[r_{i+1}, r_i]$. Hence, we can re-construct $\mathsf{Val}_{\mathcal{G}}$ by assembling well-chosen parts of the values functions of the $\mathcal{G}_{L',r_i}$ (assuming $\inf_i r_i = 0$).

This basic result will be exploited in two directions. First, we prove by induction on the number of urgent locations that all SPTGs are finitely optimal, by re-constructing $\mathsf{Val}_{\mathcal{G}}$ (as well as optimal strategies) as a $\triangleright$-concatenation of the value functions of a finite sequence of SPTGs with one more urgent locations. The base case, with only urgent locations, is solved by Proposition 4. This construction suggests a *recursive* algorithm in the spirit of [10, 18] (for non-negative prices). Second, we show that this recursion can be *avoided* (see Algorithm 2). Instead of turning locations urgent one at a time, this algorithm makes them all urgent and computes directly the sequence of SPTGs with only urgent locations. Its proof of correctness relies on the finite optimality of SPTGs and, again, on our basic result linking the values functions of $\mathcal{G}$ and games $\mathcal{G}_{L',r_i}$.

Let us formalise these constructions. Let $\mathcal{G}$ be an SPTG, let $r \in [0,1]$ be an endpoint, and let $\boldsymbol{x} = (x_\ell)_{\ell \in L}$ be a vector of rational values. Then, $\mathtt{wait}(\mathcal{G}, r, \boldsymbol{x})$ is an r-SPTG in which both players may now decide, in all non-urgent locations ℓ, to *wait* until the clock takes value r, and then to stop the game, adding the cost x_ℓ to the current cost of the play. Formally, $\mathtt{wait}(\mathcal{G}, r, \boldsymbol{x}) = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L'_f, L_u, \boldsymbol{\varphi'}, T', \pi')$ is such that $L'_f = L_f \uplus \{\ell^f \mid \ell \in L \setminus L_u\}$; for all $\ell' \in L_f$ and $\nu \in [0,r]$, $\varphi'_{\ell'}(\nu) = \varphi_{\ell'}(\nu)$, for all $\ell \in L \setminus L_u$, $\varphi'_{\ell^f}(\nu) = (r-\nu)\cdot\pi(\ell) + x_\ell$; $T' = T \cup \{(\ell, [0,r], \bot, \ell^f) \mid \ell \in L \setminus L_u\}$; for all $\delta \in T'$, $\pi'(\delta) = \pi(\delta)$ if $\delta \in T$, and $\pi'(\delta) = 0$ otherwise. Then, we let $\mathcal{G}_r = \mathtt{wait}\big(\mathcal{G}, r, (\mathsf{Val}_{\mathcal{G}}(\ell, r))_{\ell \in L}\big)$, i.e., the game obtained thanks to $\mathtt{wait}$ by letting $\boldsymbol{x}$ be the value of $\mathcal{G}$ in r. One can check that this first transformation does not alter the value of the game, for valuations before r: $\mathsf{Val}_{\mathcal{G}}(\ell, \nu) = \mathsf{Val}_{\mathcal{G}_r}(\ell, \nu)$ for all $\nu \leqslant r$.

Next, we make locations urgent. For a set $L' \subseteq L \setminus L_u$ of non-urgent locations, we let $\mathcal{G}_{L',r}$ be the SPTG obtained from $\mathcal{G}_r$ by making urgent every location ℓ of L'. Observe that, although all locations $\ell \in L'$ are now urgent in $\mathcal{G}_{L',r}$, their clones ℓ^f allow the players to wait until r. When L' is a singleton $\{\ell\}$, we write $\mathcal{G}_{\ell,r}$ instead of $\mathcal{G}_{\{\ell\},r}$. While the construction of $\mathcal{G}_r$ does not change the value of the game, introducing urgent locations *does*. Yet, we can characterise an interval $[a,r]$ on which the value functions of $\mathcal{H} = \mathcal{G}_{L',r}$ and $\mathcal{H}^+ = \mathcal{G}_{L'\cup\{\ell\},r}$ coincide, as stated by the next proposition. The interval $[a,r]$ depends on the *slopes* of the pieces of $\mathsf{Val}_{\mathcal{H}^+}$ as depicted in Figure 2: for each location ℓ of Min, the slopes of the pieces of $\mathsf{Val}_{\mathcal{H}^+}$ contained in $[a,r]$ should be $\leqslant -\pi(\ell)$ (and $\geqslant -\pi(\ell)$ when ℓ belongs to Max). It is proved by lifting optimal strategies of $\mathcal{H}^+$ into $\mathcal{H}$, and strongly relies on the determinacy result of Theorem 1:

▶ **Proposition 5.** *Let $0 \leqslant a < r \leqslant 1$, $L' \subseteq L \setminus L_u$ and $\ell \notin L' \cup L_u$ a non-urgent location of* Min *(respectively,* Max*). Assume that $\mathcal{G}_{L'\cup\{\ell\},r}$ is finitely optimal, and for all $a \leqslant \nu_1 < \nu_2 \leqslant r$*

$$\frac{\mathsf{Val}_{\mathcal{G}_{L'\cup\{\ell\},r}}(\ell,\nu_2) - \mathsf{Val}_{\mathcal{G}_{L'\cup\{\ell\},r}}(\ell,\nu_1)}{\nu_2 - \nu_1} \geqslant -\pi(\ell) \quad (\textit{respectively, } \leqslant -\pi(\ell))\,. \tag{1}$$

Then, for all $\nu \in [a,r]$ and $\ell' \in L$, $\mathsf{Val}_{\mathcal{G}_{L'\cup\{\ell\},r}}(\ell',\nu) = \mathsf{Val}_{\mathcal{G}_{L',r}}(\ell',\nu)$. Furthermore, fake-optimal NC-strategies and optimal FP-strategies in $\mathcal{G}_{L'\cup\{\ell\},r}$ are also fake-optimal and optimal over $[a,r]$ in $\mathcal{G}_{L',r}$.

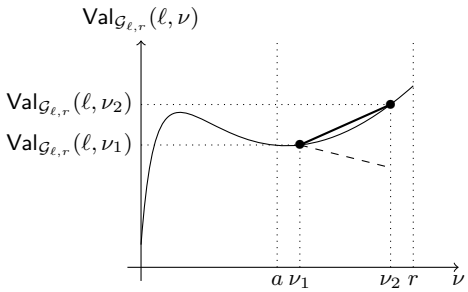


Figure 2 The condition (1) (in the case $L' = \emptyset$ and $\ell \in L_{\mathsf{Min}}$): graphically, it means that the slope between every two points of the plot in $[a, r]$ (represented with a thick line) is greater than or equal to $-\pi(\ell)$ (represented with dashed line).

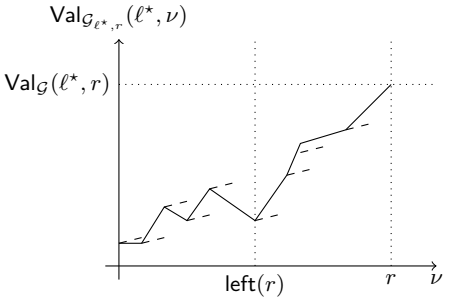


Figure 3 In this example $L' = \{\ell^\star\}$ and $\ell^\star \in L_{\mathsf{Min}}$. $\mathsf{left}(r)$ is the leftmost point such that all slopes on its right are smaller than or equal to $-\pi(\ell^\star)$ in the graph of $\mathsf{Val}_{\mathcal{G}_{\ell^\star,r}}(\ell^\star,\nu)$. Dashed lines have slope $-\pi(\ell^\star)$.

Given an SPTG $\mathcal{G}$ and some *finitely optimal* $\mathcal{G}_{L',r}$, we now characterise precisely the left endpoint of the maximal interval ending in r where the value functions of $\mathcal{G}$ and $\mathcal{G}_{L',r}$ coincide, with the operator $\mathsf{left}_{L'} : (0,1] \to [0,1]$ (or simply left, if L' is clear) defined as:

$$\mathsf{left}_{L'}(r) = \sup\{r' \leqslant r \mid \forall \ell \in L\ \forall \nu \in [r', r]\ \mathsf{Val}_{\mathcal{G}_{L',r}}(\ell,\nu) = \mathsf{Val}_{\mathcal{G}}(\ell,\nu)\}\,.$$

By continuity of the value (Theorem 1), this supremum exists and $\mathsf{Val}_{\mathcal{G}}(\ell, \mathsf{left}_{L'}(r)) = \mathsf{Val}_{\mathcal{G}_{L',r}}(\ell, \mathsf{left}_{L'}(r))$. Moreover, $\mathsf{Val}_{\mathcal{G}}(\ell)$ is a cost function on $[\mathsf{left}(r), r]$, since $\mathcal{G}_{L',r}$ is finitely optimal. However, this definition of $\mathsf{left}(r)$ is semantical. Yet, building on the ideas of Proposition 5, we can effectively compute $\mathsf{left}(r)$, given $\mathsf{Val}_{\mathcal{G}_{L',r}}$. We claim that $\mathsf{left}_{L'}(r)$ is the *minimal valuation* such that for all locations $\ell \in L' \cap L_{\mathsf{Min}}$ (respectively, $\ell \in L' \cap L_{\mathsf{Max}}$), the slopes of the affine sections of the cost function $\mathsf{Val}_{\mathcal{G}_{L',r}}(\ell)$ on $[\mathsf{left}(r), r]$ are at least (at most) $-\pi(\ell)$. Hence, $\mathsf{left}(r)$ can be obtained (see Figure 3), by inspecting iteratively, for all ℓ of Min (respectively, Max), the slopes of $\mathsf{Val}_{\mathcal{G}_{L',r}}(\ell)$, by decreasing valuations, until we find a piece with a slope $> -\pi(\ell)$ (respectively, $< -\pi(\ell)$). This enumeration of the slopes is effective as $\mathsf{Val}_{\mathcal{G}_{L',r}}$ has finitely many pieces, by hypothesis. Moreover, this guarantees that $\mathsf{left}(r) < r$. Thus, one can reconstruct $\mathsf{Val}_{\mathcal{G}}$ on $[\inf_i r_i, r_0]$ from the value functions of the (potentially infinite) sequence of games $\mathcal{G}_{L',r_0}$, $\mathcal{G}_{L',r_1}, \dots$ where $r_{i+1} = \mathsf{left}(r_i)$ for all i such that $r_i > 0$, for all possible choices of non-urgent locations L'. Next, we will define two different ways of choosing L': the former to prove finite optimality of all SPTGs, the latter to obtain an algorithm to solve them.

SPTGs are finitely optimal. To prove finite optimality of all SPTGs we reason by induction on the number of non-urgent locations and instantiate the previous results to the case where $L' = \{\ell^\star\}$ where $\ell^\star$ is a non-urgent location of *minimum price-rate* (i.e., for all $\ell \in L$, $\pi(\ell^\star) \leqslant \pi(\ell)$). Given $r_0 \in [0,1]$, we let $r_0 > r_1 > \cdots$ be the decreasing sequence of valuations such that $r_i = \mathsf{left}_{\ell^\star}(r_{i-1})$ for all $i > 0$. As explained before, we will build $\mathsf{Val}_{\mathcal{G}}$ on $[\inf_i r_i, r_0]$ from the value functions of games $\mathcal{G}_{\ell^\star,r_i}$. Assuming finite optimality of those games, this will prove that $\mathcal{G}$ is finitely optimal *under the condition* that $r_0 > r_1 > \cdots$ eventually stops, i.e., $r_i = 0$ for some i. This property is given by the next lemma, which ensures that, for all i, the owner of $\ell^\star$ has a strictly better strategy in configuration $(\ell^\star, r_{i+1})$ than waiting until r_i in location $\ell^\star$.

▶ **Lemma 6.** *If $\mathcal{G}_{\ell^\star,r_i}$ is finitely optimal for all $i \geqslant 0$, then*
(i) *if $\ell^\star \in L_{\mathsf{Min}}$ (respectively, L_{Max}), $\mathsf{Val}_{\mathcal{G}}(\ell^\star, r_{i+1}) < \mathsf{Val}_{\mathcal{G}}(\ell^\star, r_i) + (r_i - r_{i+1})\pi(\ell^\star)$ (respectively, $\mathsf{Val}_{\mathcal{G}}(\ell^\star, r_{i+1}) > \mathsf{Val}_{\mathcal{G}}(\ell^\star, r_i) + (r_i - r_{i+1})\pi(\ell^\star)$), for all i; and*
(ii) *there is $i \leqslant |\mathsf{F}_{\mathcal{G}}|^2 + 2$ such that $r_i = 0$.*

By iterating this construction, we make all locations urgent iteratively, and obtain:

▶ **Proposition 7.** *Every* SPTG $\mathcal{G}$ *is finitely optimal and for all locations ℓ, $\mathsf{Val}_{\mathcal{G}}(\ell)$ has at most $O\left((\Pi^{\mathrm{tr}}|L|^2)^{2|L|+2}\right)$ cutpoints.*

Proof. As announced, we show by induction on $n \geqslant 0$ that every r-SPTG $\mathcal{G}$ with n non-urgent locations is finitely optimal, and that the number of cutpoints of $\mathsf{Val}_{\mathcal{G}}(\ell)$ is at most $O\left((\Pi^{\mathrm{tr}}(|L_f| + n^2))^{2n+2}\right)$, which suffices to show the above bound, since $|L_f| + n^2 \leqslant |L|^2$.

The base case $n = 0$ is given by Proposition 4. Now, assume that $\mathcal{G}$ has at least one non-urgent location, and consider $\ell^\star$ one with minimum price. By induction hypothesis, all r'-SPTGs $\mathcal{G}_{\ell^\star,r'}$ are finitely optimal for all $r' \in [0, r]$. Let $r_0 > r_1 > \cdots$ be the decreasing sequence defined by $r_0 = r$ and $r_i = \mathsf{left}_{\ell^\star}(r_{i-1})$ for all $i \geqslant 1$. By Lemma 6, there exists $j \leqslant |\mathsf{F}_{\mathcal{G}}|^2 + 2$ such that $r_j = 0$. Moreover, for all $0 < i \leqslant j$, $\mathsf{Val}_{\mathcal{G}} = \mathsf{Val}_{\mathcal{G}_{\ell^\star,r_{i-1}}}$ on $[r_i, r_{i-1}]$ by definition of $r_i = \mathsf{left}_{\ell^\star}(r_{i-1})$, so that $\mathsf{Val}_{\mathcal{G}}(\ell)$ is a cost function on this interval, for all ℓ, and the number of cutpoints on this interval is bounded by $O\left((\Pi^{\mathrm{tr}}(|L_f| + (n-1)^2 + n))^{2(n-1)+2}\right) = O\left((\Pi^{\mathrm{tr}}(|L_f| + n^2))^{2(n-1)+2}\right)$ by induction hypothesis (notice that maximal transition prices are the same in $\mathcal{G}$ and $\mathcal{G}_{\ell^\star,r_{i-1}}$, but that we add n more final locations in $\mathcal{G}_{\ell^\star,r_{i-1}}$). Adding the cutpoint 1, summing over i from 0 to $j \leqslant |\mathsf{F}_{\mathcal{G}}|^2 + 2$, and observing that $|\mathsf{F}_{\mathcal{G}}| \leqslant 2\Pi^{\mathrm{tr}}|L_f|$, we bound the number of cutpoints of $\mathsf{Val}_{\mathcal{G}}(\ell)$ by $O\left((\Pi^{\mathrm{tr}}(|L_f| + n^2))^{2n+2}\right)$. Finally, we can reconstruct fake-optimal and optimal strategies in $\mathcal{G}$ from the from fake-optimal and optimal strategies of $\mathcal{G}_{\ell^\star,r_i}$. ◀

Computing the value functions. The finite optimality of SPTGs allows us to compute the value functions. The proof of Proposition 7 suggests a *recursive* algorithm to do so: from an SPTG $\mathcal{G}$ with minimal non-urgent location $\ell^\star$, solve recursively $\mathcal{G}_{\ell^\star,1}$, $\mathcal{G}_{\ell^\star,\mathsf{left}(1)}$, $\mathcal{G}_{\ell^\star,\mathsf{left}(\mathsf{left}(1))}$, *etc.* handling the base case where all locations are urgent with Algorithm 1. While our results above show that this is correct and terminates, we propose instead to solve – without the need for recursion – the sequence of games $\mathcal{G}_{L\setminus L_u,1}$, $\mathcal{G}_{L\setminus L_u,\mathsf{left}(1)}, \ldots$ i.e., *making all locations urgent at once*. Again, the arguments given above prove that this scheme is *correct*, but the key argument of Lemma 6 that ensures *termination* can not be applied in this case. Instead, we rely on the following lemma, stating, that there will be at least one cutpoint of $\mathsf{Val}_{\mathcal{G}}$ in each interval $[\mathsf{left}(r), r]$. Observe that this lemma relies on the fact that $\mathcal{G}$ is finitely optimal, hence the need to first prove this fact independently with the sequence $\mathcal{G}_{\ell^\star,1}$, $\mathcal{G}_{\ell^\star,\mathsf{left}(1)}$, $\mathcal{G}_{\ell^\star,\mathsf{left}(\mathsf{left}(1))}, \ldots$ Termination then follows from the fact that $\mathsf{Val}_{\mathcal{G}}$ has finitely many cutpoints by finite optimality.

▶ **Lemma 8.** *Let $r_0 \in (0, 1]$ such that $\mathcal{G}_{L',r_0}$ is finitely optimal. Suppose that $r_1 = \mathsf{left}_{L'}(r_0) > 0$, and let $r_2 = \mathsf{left}_{L'}(r_1)$. There exists $r' \in [r_2, r_1)$ and $\ell \in L'$ such that*
(i) *$\mathsf{Val}_{\mathcal{G}}(\ell)$ is affine on $[r', r_1]$, of slope equal to $-\pi(\ell)$, and*
(ii) *$\mathsf{Val}_{\mathcal{G}}(\ell, r_1) \neq \mathsf{Val}_{\mathcal{G}}(\ell, r_0) + \pi(\ell)(r_0 - r_1)$.*
As a consequence, $\mathsf{Val}_{\mathcal{G}}(\ell)$ has a cutpoint in $[r_1, r_0)$.

Algorithm 2 implements these ideas. Each iteration of the **while** loop computes a new game in the sequence $\mathcal{G}_{L\setminus L_u,1}$, $\mathcal{G}_{L\setminus L_u,\mathsf{left}(1)}, \ldots$ described above; solves it thanks to `solveInstant`; and thus computes a new portion of $\mathsf{Val}_{\mathcal{G}}$ on an interval on the left of the current point $r \in [0, 1]$. More precisely, the vector $(\mathsf{Val}_{\mathcal{G}}(\ell, 1))_{\ell \in L}$ is first computed in line 1.

Algorithm 2: `solve`$(\mathcal{G})$

Input: SPTG $\mathcal{G} = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_f, L_u, \boldsymbol{\varphi}, \Delta, \pi)$

1 $\boldsymbol{f} = (f_\ell)_{\ell \in L} :=$ `solveInstant`$(\mathcal{G}, 1)$ /* $f_\ell \colon \{1\} \to \overline{\mathbf{R}}$ */
2 $r := 1$
3 **while** $0 < r$ **do** /* Invariant: $f_\ell \colon [r, 1] \to \overline{\mathbf{R}}$ */
4 $\quad \mathcal{G}' :=$ `wait`$(\mathcal{G}, r, \boldsymbol{f}(r))$ /* r-SPTG $\mathcal{G}' = (L_{\mathsf{Min}}, L_{\mathsf{Max}}, L'_f, L'_u, \boldsymbol{\varphi}', T', \pi')$ */
5 $\quad L'_u := L'_u \cup L$ /* every location is made urgent */
6 $\quad b := r$
7 $\quad$ **repeat** /* Invariant: $f_\ell \colon [b, 1] \to \overline{\mathbf{R}}$ */
8 $\quad\quad a := \max(\mathsf{PossCP}_{\mathcal{G}'} \cap [0, b))$
9 $\quad\quad \boldsymbol{x} = (x_\ell)_{\ell \in L} :=$ `solveInstant`$(\mathcal{G}', a)$ /* $x_\ell = \mathsf{Val}_{\mathcal{G}'}(\ell, a)$ */
10 $\quad\quad$ **if** $\forall \ell \in L_{\mathsf{Min}}\ \frac{f_\ell(b) - x_\ell}{b-a} \leqslant -\pi(\ell) \wedge \forall \ell \in L_{\mathsf{Max}}\ \frac{f_\ell(b) - x_\ell}{b-a} \geqslant -\pi(\ell)$ **then**
11 $\quad\quad\quad$ **foreach** $\ell \in L$ **do** $f_\ell := \left(\nu \in [a, b] \mapsto f_\ell(b) + (\nu - b)\frac{f_\ell(b) - x_\ell}{b-a}\right) \rhd f_\ell$
12 $\quad\quad\quad b := a\,;\ stop := false$
13 $\quad\quad$ **else** $stop := true$
14 $\quad$ **until** $b = 0$ *or* $stop$
15 $\quad r := b$
16 **return** $\boldsymbol{f}$

Then, the algorithm enters the **while** loop, and the game $\mathcal{G}'$ obtained when reaching line 6 is $\mathcal{G}_{L \setminus L_u, 1}$. Then, the algorithm enters the **repeat** loop to analyse this game. Instead of building the whole value function of $\mathcal{G}'$, Algorithm 2 builds only the parts of $\mathsf{Val}_{\mathcal{G}'}$ that coincide with $\mathsf{Val}_{\mathcal{G}}$. It proceeds by enumerating the possible cutpoints a of $\mathsf{Val}_{\mathcal{G}'}$, starting in r, by decreasing valuations (line 8), and computes the value of $\mathsf{Val}_{\mathcal{G}'}$ in each cutpoint thanks to `solveInstant` (line 9), which yields a new piece of $\mathsf{Val}_{\mathcal{G}'}$. Then, the **if** in line 10 checks whether this new piece coincides with $\mathsf{Val}_{\mathcal{G}}$, using the condition given by Proposition 5. If it is the case, the piece of $\mathsf{Val}_{\mathcal{G}'}$ is added to f_ℓ (line 11); **repeat** is stopped otherwise. When exiting the **repeat** loop, variable b has value $\mathsf{left}(1)$. Hence, at the next iteration of the **while** loop, $\mathcal{G}' = \mathcal{G}_{L \setminus L_u, \mathsf{left}(1)}$ when reaching line 6. By continuing this reasoning inductively, one concludes that the successive iterations of the **while** loop compute the sequence $\mathcal{G}_{L \setminus L_u, 1}$, $\mathcal{G}_{L \setminus L_u, \mathsf{left}(1)}, \ldots$ as announced, and rebuilds $\mathsf{Val}_{\mathcal{G}}$ from them. Termination in exponential time is ensured by Lemma 8: each iteration of the **while** loop discovers at least one new cutpoint of $\mathsf{Val}_{\mathcal{G}}$, and there are at most exponentially many (note that a tighter bound on this number of cutpoints would entail a better complexity of our algorithm).

▶ **Example 9.** Let us briefly sketch the execution of Algorithm 2 on the SPTG in Figure 1. During the first iteration of the **while** loop, the algorithm computes the correct value functions until the cutpoint $\frac{3}{4}$: in the *repeat* loop, at first $a = 9/10$ but the slope in ℓ_1 is smaller than the slope that would be granted by waiting, as depicted in Figure 1. Then, $a = 3/4$ where the algorithm gives a slope of value -16 in ℓ_2 while the cost of this location of Max is -14. During the first iteration of the **while** loop, the inner **repeat** loop thus ends with $r = 3/4$. The next iterations of the **while** loop end with $r = \frac{1}{2}$ (because ℓ_1 does not pass the test in line 10); $r = \frac{1}{4}$ (because of ℓ_2) and finally with $r = 0$, giving us the value functions on the entire interval $[0, 1]$.

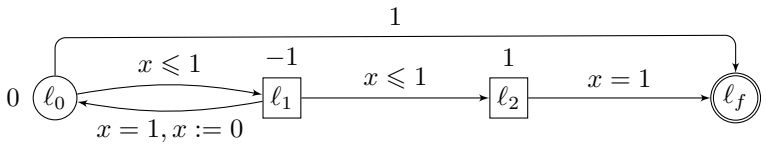


Figure 4 A PTG where the number of resets in optimal plays can not be bounded a priori.

5 Beyond SPTGs

In [10, 18, 16], *general* PTGs with *non-negative prices* are solved by reducing them to a finite sequence of SPTGs, by eliminating guards and resets. It is thus natural to try and adapt these techniques to our general case, in which case Algorithm 2 would allow us to solve *general* PTG*s with arbitrary costs*. Let us explain why it is not (completely) the case. The technique used to remove guards from PTGs consists in enhancing the locations with regions while keeping an equivalent game. This technique *can* be adapted to arbitrary weights.

The technique to handle resets, however, consists in *bounding* the number of clock resets that can occur in each play following an optimal strategy of Min or Max. Then, the PTG can be *unfolded* into a *reset-acyclic* PTG with the same value. By reset-acyclic, we mean that no cycle in the configuration graph visits a transition with a reset. This reset-acyclic PTG can be decomposed into a finite number of components that contain no reset and are linked by transitions with resets. These components can be solved iteratively, from the bottom to the top, turning them into SPTGs. Thus, if we *assume* that the PTGs we are given as input *are* reset-acyclic, we can solve them in *exponential time*, and show that their value functions are cost functions with at most exponentially many cutpoints, using our techniques. Unfortunately, the arguments to bound the number of resets do not hold for arbitrary costs, as shown by the PTG in Figure 4. We claim that $\mathsf{Val}(\ell_0) = 0$; that Min has no optimal strategy, but a family of ε-optimal strategies $\sigma^{\varepsilon}_{\mathsf{Min}}$ each with value ε; and that each $\sigma^{\varepsilon}_{\mathsf{Min}}$ requires *memory whose size depends on* ε and might *yield a play visiting at least* $1/\varepsilon$ *times the reset* between ℓ_0 and ℓ_1 (hence the number of resets can not be bounded). For all $\varepsilon > 0$, $\sigma^{\varepsilon}_{\mathsf{Min}}$ consists in: waiting $1 - \varepsilon$ time units in ℓ_0, then going to ℓ_1 during the $\lceil 1/\varepsilon \rceil$ first visits to ℓ_0; and to go directly to ℓ_f afterwards. Against $\sigma^{\varepsilon}_{\mathsf{Min}}$, Max has two possible choices:

- **(i)** either wait 0 time unit in ℓ_1, wait ε time units in ℓ_2, then reach ℓ_f; or
- **(ii)** wait ε time unit in ℓ_1 then force the cycle by going back to ℓ_0 and wait for Min's next move.

Thus, all plays according to $\sigma^{\varepsilon}_{\mathsf{Min}}$ will visit a sequence of locations which is either of the form $\ell_0(\ell_1\ell_0)^k\ell_1\ell_2\ell_f$, with $0 \leqslant k < \lceil 1/\varepsilon \rceil$; or of the form $\ell_0(\ell_1\ell_0)^{\lceil \frac{1}{\varepsilon} \rceil}\ell_f$. In the former case, the cost of the play will be $-k\varepsilon + 0 + \varepsilon = -(k-1)\varepsilon \leqslant \varepsilon$; in the latter, $-\varepsilon(\lceil 1/\varepsilon \rceil) + 1 \leqslant 0$. This shows that $\mathsf{Val}(\ell_0) = 0$, but there is no optimal strategy as none of these strategies allow one to guarantee a cost of 0 (neither does the strategy that waits 1 time unit in ℓ_0).

However, we may apply the result on reset-acyclic PTGs to obtain:

▶ **Theorem 10.** *The value functions of all one-clock* PTG*s are cost functions with at most exponentially many cutpoints.*

Proof. Let $\mathcal{G}$ be a one-clock PTG. Let us replace all transitions $(\ell, g, \top, \ell')$ resetting the clock by $(\ell, g, \bot, \ell'')$, where ℓ'' is a new final location with $\varphi_{\ell''} = \mathsf{Val}_{\mathcal{G}}(\ell, 0)$ – observe that $\mathsf{Val}_{\mathcal{G}}(\ell, 0)$ exists even if we can not compute it, so this transformation is well-defined. This yields a reset-acyclic PTG $\mathcal{G}'$ such that $\mathsf{Val}_{\mathcal{G}'} = \mathsf{Val}_{\mathcal{G}}$. ◀

References

1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

4 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

5 J. Berendsen, T. Chen, and D. Jansen. Undecidability of cost-bounded reachability in priced probabilistic timed automata. In *Theory and Applications of Models of Computation*, volume 5532 of *LNCS*, pages 128–137. Springer, 2009.

6 Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.

7 Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

8 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.

9 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. Research Report LSV-14-12, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2014. 24 pages.

10 Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in one-clock priced timed games. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

11 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proceedings of the Third international conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.

12 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. Simple priced timed games are not that simple. Research Report 1507.03786, arXiv, July 2015.

13 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. To reach or not to reach? Efficient algorithms for total-payoff games. In Luca Aceto and David de Frutos Escrig, editors, *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPIcs*, pages 297–310. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, September 2015.

14 Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding Negative Prices to Priced Timed Games. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'13)*, volume 8704 of *Lecture Notes in Computer Science*, pages 560–575. Springer, 2014.

15 D. Gale and F. M. Stewart. Infinite games with perfect information. In *Contributions to the theory of games, vol. 2. Annals of Mathematical Studies*, volume 28 of *Lecture Notes in Computer Science*, pages 245–266. Princeton University Press., 1953.

16 Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.

17 Peter J. Ramadge and W. Murray Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77(1), pages 81–98, 1989.

18 Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In *Proceedings of the 9th Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, volume 57 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–46, 2011.

Quantitative Games under Failures*

Thomas Brihaye[1], Gilles Geeraerts[2], Axel Haddad[1], Benjamin Monmege[3], Guillermo A. Pérez[†2], and Gabriel Renault[1]

1 **Université de Mons, Belgium**
{thomas.brihaye,axel.haddad,gabriel.renault}@umons.ac.be
2 **Université libre de Bruxelles, Belgium**
{gigeerae,gperezme}@ulb.ac.be
3 **LIF, Aix-Marseille Université, CNRS, France**
benjamin.monmege@lif.univ-mrs.fr

Abstract

We study a generalisation of sabotage games, a model of dynamic network games introduced by van Benthem [16]. The original definition of the game is inherently finite and therefore does not allow one to model infinite processes. We propose an extension of the sabotage games in which the first player (Runner) traverses an arena with dynamic weights determined by the second player (Saboteur). In our model of *quantitative sabotage games*, Saboteur is now given a budget that he can distribute amongst the edges of the graph, whilst Runner attempts to minimise the quantity of budget witnessed while completing his task. We show that, on the one hand, for most of the classical cost functions considered in the literature, the problem of determining if Runner has a strategy to ensure a cost below some threshold is EXPTIME-complete. On the other hand, if the budget of Saboteur is fixed a priori, then the problem is in PTIME for most cost functions. Finally, we show that restricting the dynamics of the game also leads to better complexity.

1998 ACM Subject Classification F.1.1 Automata, D.2.4 Formal methods

Keywords and phrases Quantitative games, Verification, Synthesis, Game theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.293

1 Introduction

Two-player games played on graphs are nowadays a well-established model for systems where two antagonistic agents interact. In particular, they allow one to perform controller synthesis [1], when one of the players models the controller, and the second plays the role of an evil environment. Quantitative generalisations (played on weighted graphs) of these models have attracted much attention in the last decades [5, 8, 3] as they allow for a finer analysis of those systems.

In this setting, most results assume that the arena (i.e., the graph) on which the game is played does not change during the game. There are however many situations where this restriction is not natural, at least from a modelling point of view. For instance, Grüner *et al.* [7] model connectivity problems in *dynamic* networks (i.e., subject to failure and restoration) using a variant of *sabotage games* – a model originally proposed by van Benthem [16] – to model *reachability problems* in a network prone to errors. A sabotage game is played on a directed graph, and starts with a token in an initial vertex. Then, Runner and Saboteur (the

* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under Grant Agreement n° 601148 (CASSTING).
† Author supported by F.R.S.-FNRS fellowship.

two players of the game) play in alternation: Runner moves the token along one edge and Saboteur is allowed to remove one edge. Runner wins the game if he reaches a target set of vertices. In [12], it is shown that deciding the existence of a winning strategy for Runner is PSPACE-complete.

In those sabotage games, errors are regarded as unrecoverable failures. In practice, this hypothesis might be too strong. Instead, one might want to model the fact that certain uncontrollable events incur additional costs (modelling delays, resource usage...), and look for strategies that allow one to fulfil the game objective *at a minimal cost*, whatever the occurrence of uncontrollable events. For instance, if the graph models a railway network, the failure of a track will eventually be fixed, and, in the meantime, trains might be slowed down on the faulty portion or diverted, creating delays in the journeys. It is thus natural to consider *quantitative* extensions of sabotage games, where Saboteur controls the price of the actions in the game. This is the aim of the present paper.

More precisely, we extend sabotage games in two directions. First, we consider games played on *weighted* graphs. Saboteur is allotted an integral budget B that he can distribute (dividing it into integral parts) on the edges of the graph, thereby setting their weights. At each turn, Saboteur can change this distribution by moving k units of budget from an edge to another edge (for simplicity, we restrict ourselves to $k = 1$ but our results hold for any k). Second, we relax the inherent finiteness of sabotage games (all edges will eventually be deleted), and consider infinite horizon games (i.e., plays are now infinite). In this setting, the goal of Runner is to minimise the cost defined by the sequence of weights of edges visited, with respect to some fixed cost function (Inf, Sup, LimInf, LimSup, average or discounted-sum), while Saboteur attempts to maximise the same cost. We call these games *quantitative sabotage games* (QSG, for short).

Let us briefly sketch one potential application of our model, showing that they are useful to perform synthesis in a dynamic environment. Our application is borrowed from Suzuki and Yamashita [17] who have considered the problem of *motion planning* of multiple mobile robots that interact in a finite space. In essence, each robot executes a "Look-Compute-Move" cycle and should realise some specification (that we could specify using LTL, for instance). For simplicity, assume that at every observation (Look) phase, at most one other robot has moved. Clearly every motion phase (Move) will require different amounts of time and energy depending on the location of the other robots. We can model the interaction of each individual robot against all others using a QSG where Runner is one robot, Saboteur is the coalition of all other robots, and the budget is equal to the number of robots minus 1. This model allows one to answer meaningful questions such as '*what is, in the worst case, the average delay the robot incurs because of the dynamics of the system?*', or '*what is the average amount of additional energy required because of the movements of the other robots?*' using appropriate cost functions.

As a second motivational example, let us recall the motivation of the original Sabotage Game: consider a situation in which you need to find your way between two cities within a railway network where a malevolent demon starts cancelling connections? This is called the *real Travelling Salesman Problem* by van Benthem [16]. However, in real life, railway companies have contracts with infrastructure companies which ensure that failures in the railway network are repaired withing a given amount of time (e.g. a service-level agreement). In this case, it is better to consider delays instead of absolute failures in the network. Further, salesmen do not usually have one single trip in their whole carriers. For modelling purposes, one can in fact assume they never stop travelling. In this setting, QSGs can be used to answer the question: '*what is, in the worst case, the average delay time incurred by the salesman*'?

Table 1 Complexity results for quantitative sabotage games.

	QSG	static QSG	fixed budget QSG
Inf, LimInf	$\in$ EXPTIME	$\in$ PTIME	$\in$ PTIME
Sup, LimSup, Avg	EXPTIME-c	coNP-c	$\in$ PTIME
DS	EXPTIME-c	coNP-c	$\in$ NP $\cap$ coNP

Our model can be used to treat the same questions for other networks and not just railway networks.

Related Works & Contributions. Variations of the original sabotage games have been considered by students of van Benthem. In [11], the authors have considered changing the *reachability objective* of Runner to a *safety objective*, and proved it is PSPACE-complete as well. They also consider a co-operative variation of the game which, not surprisingly, leads to a lower complexity: NL-complete. In [14], an asymmetric imperfect information version of the game is studied—albeit, under the guise of the well-known parlor game *Scotland Yard*—and shown to be PSPACE-complete. We remark that although the latter version of sabotage games already includes some sort of dynamicity in the form of the *Scotland Yard* team moving their pawns on the board, both of these studies still focus on inherently finite versions of the game.

We establish that QSGs are EXPTIME-complete in general. Our approach is to prove the result for a very weak problem on QSGs, called the *safety problem*, that asks whether Runner can avoid *ad vitam æternam* edges with non-zero budget on it. We remark that although the safety problem is related to cops and robbers games [1, 6], we were not able to find EXPTIME-hard variants that reduce easily into our formalism. The general problem being EXPTIME-complete, we consider the case where the budget is fixed instead of left as an input of the problem (see Corollary 2). We also consider restricting the behaviour of Saboteur and define a variation of our QSGs in which Saboteur is only allowed to choose an initial distribution of weights but has to commit to it once he has fixed it. We call this the *static* version of the game. For both restrictions, we show that tractable algorithms exist for some of the cost functions we consider. A summary of the complexity results we establish in this work is shown in Table 1. In Section 6, we comment on several implications of the complexity bounds proved in this work.

Some proofs and technical details may be found in the long version [2].

2 Quantitative sabotage games

Let us now formally define quantitative sabotage games (QSG). We start with the definition of the cost functions we will consider, then give the syntax and semantics of QSG.

Cost functions. A *cost function* $f\colon \mathbb{Q}^\omega \to \mathbb{R}$ associates a real number to a sequence of rationals $u = (u_i)_{i\geqslant 0} \in \mathbb{Q}^\omega$. The six classical cost functions that we consider are

- $\mathsf{Inf}(u) = \inf\{u_i \mid i \geqslant 0\}$;
- $\mathsf{Sup}(u) = \sup\{u_i \mid i \geqslant 0\}$;
- $\mathsf{LimInf}(u) = \liminf_{n\to\infty}\{u_i \mid i \geqslant n\}$;
- $\mathsf{LimSup}(u) = \limsup_{n\to\infty}\{u_i \mid i \geqslant n\}$;
- $\mathsf{Avg}(u) = \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} u_i$, which stands for the average cost (also called *mean-payoff* in the literature); and
- $\mathsf{DS}_\lambda(u) = \sum_{i=0}^{\infty} \lambda^i \cdot u_i$, (with $0 < \lambda < 1$), stands for discounted-sum.

In the following, we let $\mathsf{DS} = \{\mathsf{DS}_\lambda \mid 0 < \lambda < 1\}$.

Syntax. As sketched in the introduction, *quantitative sabotage games* are played by Runner and Saboteur on a directed weighted graph, called the *arena*. A play alternates between Runner moving the token along the edges and Saboteur modifying the weights. We consider that Saboteur has a fixed integer budget B that he can distribute on edges, thereby setting their weights (which must be integer values). Formally, for a finite set E and a budget $B \in \mathbb{N}$, $\Delta(E, B)$ denotes the set of all *distributions* of budget B on E, where a distribution is a function $\delta \colon E \to \{0, 1, \dots, B\}$ such that $\sum_{e \in E} \delta(e) \leqslant B$ (the last constraint is an inequality since the whole budget need not be distributed on E). Then, a *quantitative sabotage game* is a tuple $\mathcal{G} = (V, E, B, v_I, \delta_I, f)$, where (V, E) is a directed graph, $B \in \mathbb{N}$ is the budget of the game, $v_I \in V$ is the initial vertex, $\delta_I \in \Delta(E, B)$ is the initial distribution of the budget, and f is a cost function. We assume, without loss of generality, that there are no deadlocks in (V, E), i.e., for all $v \in V$, there is $v' \in V$ such that $(v, v') \in E$. In the following, we may alternatively write $\Delta(\mathcal{G})$ for $\Delta(E, B)$ when $\mathcal{G}$ is a QSG with set of edges E and budget B.

Semantics. To define the semantics of a QSG $\mathcal{G}$, we first formalise the possible redistributions of the budget by Saboteur. We choose to restrict them, reflecting some physical constraints: Saboteur can move at most one unit of weight in-between two edges. For $\delta, \delta' \in \Delta(\mathcal{G})$, we say that δ' is a *valid redistribution* from δ, noted $\delta \triangleright \delta'$, if and only if there are $e_1, e_2 \in E$ such that $\delta'(e_1) \in \{\delta(e_1), \delta(e_1) - 1\}$, $\delta'(e_2) \in \{\delta(e_2), \delta(e_2) + 1\}$, and for all other edges $e \notin \{e_1, e_2\}$, $\delta'(e) = \delta(e)$. Then, a *play* in a QSG $\mathcal{G} = (V, E, B, v_I, \delta_I, f)$ is an infinite sequence $\pi = v_0\delta_0 v_1\delta_1 \cdots$ alternating vertices $v_i \in V$ and budget distributions $\delta_i \in \Delta(\mathcal{G})$ such that

(i) $v_0 = v_I$;
(ii) $\delta_0 = \delta_I$; and
(iii) for all $i \geqslant 0$: $(v_i, v_{i+1}) \in E$, and $\delta_i \triangleright \delta_{i+1}$.

Let $\mathrm{Prefs}_\Delta(\mathcal{G})$ denote the set of prefixes of plays ending in a budget distribution, and $\mathrm{Prefs}_V(\mathcal{G})$ the set of prefixes of length at least 2 ending in a vertex. We abuse notations and lift cost functions f to plays letting $f(v_0\delta_0 v_1\delta_1 \cdots) = f(\delta_0(v_0, v_1)\delta_1(v_1, v_2) \cdots)$. A *strategy* of Runner is a mapping $\rho \colon \mathrm{Prefs}_\Delta(\mathcal{G}) \to V$ such that $(v_n, \rho(\pi)) \in E$ for all $\pi = v_0\delta_0 \cdots v_n\delta_n \in \mathrm{Prefs}_\Delta(\mathcal{G})$. A strategy of Saboteur is a mapping $\sigma \colon \mathrm{Prefs}_V(\mathcal{G}) \to \Delta(\mathcal{G})$ such that $\delta_{n-1} \triangleright \sigma(\pi)$ for all $\pi = v_0\delta_0 \cdots v_{n-1}\delta_{n-1}v_n \in \mathrm{Prefs}_V(\mathcal{G})$. We denote by $\Sigma_{\mathrm{Run}}(\mathcal{G})$ (respectively, $\Sigma_{\mathrm{Sab}}(\mathcal{G})$) the set of all strategies of Runner (respectively, Saboteur). A pair of strategies (ρ, σ) of Runner and Saboteur defines a unique play $\pi_{\rho,\sigma} = v_0\delta_0 v_1\delta_1 \cdots$ such that for all $i \geqslant 0$:

(i) $v_{i+1} = \rho(v_0\delta_0 \cdots v_i\delta_i)$; and
(ii) $\delta_{i+1} = \sigma(v_0\delta_0 \cdots v_i\delta_i v_{i+1})$.

Values and determinacy. We are interested in computing the best value that each player can guarantee no matter how the other player plays. To reflect this, we define two values of a QSG $\mathcal{G}$: the superior value (modelling the best value for Runner)as $\overline{\mathbf{Val}}(\mathcal{G}) := \sup_{\sigma \in \Sigma_{\mathrm{Sab}}(\mathcal{G})} \inf_{\rho \in \Sigma_{\mathrm{Run}}(\mathcal{G})} f(\pi_{\rho,\sigma})$, and the inferior value (modelling the best value for Saboteur) as $\underline{\mathbf{Val}}(\mathcal{G}) := \inf_{\rho \in \Sigma_{\mathrm{Run}}(\mathcal{G})} \sup_{\sigma \in \Sigma_{\mathrm{Sab}}(\mathcal{G})} f(\pi_{\rho,\sigma})$. It is folklore to prove that $\underline{\mathbf{Val}}(\mathcal{G}) \leqslant \overline{\mathbf{Val}}(\mathcal{G})$. Indeed, for the previously mentioned cost functions, we can prove that QSGs are determined, i.e., that $\underline{\mathbf{Val}}(\mathcal{G}) = \overline{\mathbf{Val}}(\mathcal{G})$ for all QSGs $\mathcal{G}$. This can be formally proved by encoding a QSG $\mathcal{G}$ into a quantitative two-player game $[\![\mathcal{G}]\!]$ (whose vertices contain both vertices of $\mathcal{G}$ and budget distributions), and then using classical Martin's determinacy theorem [13]. $\underline{\mathbf{Val}}(\mathcal{G}) = \overline{\mathbf{Val}}(\mathcal{G})$ is henceforth called the *value of* $\mathcal{G}$, and denoted by $\mathbf{Val}(\mathcal{G})$.

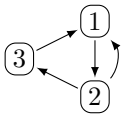


Figure 1 A QSG.

Example. Consider the simple QSG $\mathcal{G}$ in Figure 1, where the budget of Saboteur is $B = 4$, and the cost function is Avg. We claim that whatever the initial configuration, $\mathbf{Val}(\mathcal{G}) = 2$. Indeed, consider the strategy of Saboteur that consists in eventually putting all the budget on the edge $(①,②)$ (i.e., letting $\delta(①,②) = 4$ and $\delta(e) = 0$ for all other edges e), and then playing as follows: whenever Runner reaches ②, move one unit of budget from $(①,②)$ to $(②,③)$; if Runner moves to ③, move the unit of budget from $(②,③)$ to $(③,①)$; and when Runner moves back to ①, move all the budget back on $(①,②)$, by consuming one unit either from $(②,③)$ or from $(③,①)$. Let us call this strategy $\overline{\sigma}$. Since we consider the average cost, only the long-term behaviour of Runner is relevant to compute the cost of a play. So, as soon as Saboteur has managed to reach a distribution δ such that $\delta(①,②) = 4$, the only choices for Runner each time he visits ① are either to visit the ①–②–③–① cycle, or the ①–②–① cycle. In the former case, Runner traverses 3 edges and pays $4 + 1 + 1 = 6$, hence an average cost of $\frac{6}{3} = 2$ for this cycle. In the latter, he pays an average of $\frac{4+0}{2} = 2$ for the cycle. Hence, whatever the strategy ρ of Runner, we have $\mathsf{Avg}(\pi_{\overline{\sigma},\rho}) = 2$, which proves that $\underline{\mathbf{Val}}(\mathcal{G}) \geqslant 2$. One can check that the strategy $\overline{\rho}$ of Runner consisting in always playing the ①–②–③–① cycle indeed guarantees cost 2, proving that $\overline{\mathbf{Val}}(\mathcal{G}) \leqslant 2$. This proves that the value $\mathbf{Val}(\mathcal{G})$ of the game is 2.

3 Solving quantitative sabotage games

Given a QSG, our main objective is to determine whether Runner can play in such a way that he will ensure a cost at most T, no matter how Saboteur plays, and where T is a given threshold. This amounts to determining whether $\mathbf{Val}(\mathcal{G}) \leqslant T$. Thus, for a cost function f, the Threshold problem with cost function f consists in determining whether $\mathbf{Val}(\mathcal{G}) \leqslant T$, given a QSG $\mathcal{G}$ with cost function f and a non-negative threshold T. When $f = \mathsf{DS}$, we assume that the discount factor λ is part of the input. If we want it to be a parameter of the problem (and not a part of the input), we consider $f = \mathsf{DS}_\lambda$. Our main contribution is to characterise the complexity of the threshold problem for all the cost functions introduced before, as summarised in the following theorem:

▶ **Theorem 1.** *For cost functions* Sup, LimSup, Avg, DS *and* DS_λ, *the threshold problem over* QSG*s is* EXPTIME*-complete; for* Inf *and* LimInf, *it is in* EXPTIME.

For all cost functions, the EXPTIME membership is established by using the encoding of a QSG $\mathcal{G}$ into a classical quantitative two-player game $[\![\mathcal{G}]\!]$ which is played on a weighted graph, whose vertices are the configurations of the sabotage game, i.e., a tuple containing the current vertex, the last crossed edge and the current weight distribution, and whose weights are in $\{0, \dots, B\}$ (describing how much runner pays by moving from one configuration to another). Notice that $\Delta(\mathcal{G})$ has size at most $(B+1)^{|E|}$, since every distribution is a mapping of $E \to \{0, 1, \dots, B\}$. Hence, we see that the game $[\![\mathcal{G}]\!]$ has a number of vertices at most exponential with respect to $|V|$, and polynomial with respect to B (which, being given in binary, can be exponential in the size of the input of the problem). Using results from [18, 3, 1], we know that we can compute in pseudo-polynomial time the value of the quantitative game $[\![\mathcal{G}]\!]$ for all the cost functions cited in the theorem: here, pseudo-polynomial means polynomial with respect to the number of vertices and edges of $[\![\mathcal{G}]\!]$ (which is exponential with respect to $|V|$), and polynomial with respect to the greatest weight in absolute value, here B (which is also exponential with respect to $|V|$). Thus we obtain the exponential time upper bound

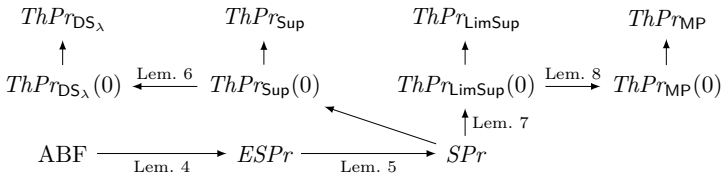


Figure 2 Reductions used in this section. We denote by $ThPr_f$ (respectively, $ThPr_f(0)$) the threshold problem (respectively, the sub-problem of the threshold problem where threshold is 0) for QSGs with cost function f. Non-trivial reductions are labelled with the corresponding lemma stated in this section.

announced in the theorem. Note that for DS_λ, pseudo-polynomial also means polynomial in the value of the denominator of λ.[1]

When the budget B is fixed, i.e., when it is a parameter of the problem and not one of the inputs, the explanation above can be adapted to prove that the problem is solvable in polynomial time for all but the DS_λ cost functions. Indeed, we can refine our analysis of the size of $\Delta(\mathcal{G})$. A budget distribution can also be encoded as a mapping $\gamma\colon \{1,\dots,B\} \to E$ where we consider the budget as a set of indexed pebbles: such a mapping represents the distribution δ defined by $\delta(e) = |\gamma^{-1}(e)|$. This encoding shows that $\Delta(\mathcal{G})$ has size at most $|E|^B$, which is polynomial in $|E|$. For the discounted sum, the role of λ in the complexity stays the same, causing an NP ∩ coNP and pseudo-polynomial complexity: this blow-up disappears if λ is a parameter of the problem. In the overall, we obtain:

▶ **Corollary 2.** *For cost functions* Inf, Sup, LimInf, LimSup, Avg, DS_λ, *and for fixed budget B, the threshold problem for* QSG*s is in* PTIME*; for* DS *(where λ is an input), it is in* NP ∩ coNP *and can be solved in pseudo-polynomial time.*

The rest of this section is devoted to the proof of EXPTIME-hardness in Theorem 1 for cost functions Sup, LimSup, Avg and DS_λ (this implies EXPTIME-hardness for DS too). Our gold-standard problem for EXPTIME-hardness is the *alternating Boolean formula* (ABF) problem, introduced by Stockmeyer and Chandra in [15]. Our proof consists of a sequence of reductions from this problem, as depicted in Figure 2. First, we show a reduction to the threshold problem for Sup cost function when the threshold is 0 and **the initial distribution is empty** (i.e., no budget on any edge), on QSGs extended with *safe edges* and *final vertices* (in order to make the reduction more readable). Notice that this problem amounts to determining whether Runner has a strategy to avoid crossing an edge with non-zero budget, therefore we refer to this problem as the *extended safety problem* (*ESPr*). Our next step is to encode safe edges and final vertices into (non-extended) QSGs with gadgets of polynomial size, therefore proving that the *safety problem* (*SPr*) is itself EXPTIME-hard: *SPr* is a special case of the threshold problem $ThPr_{\mathsf{Sup}}(0)$ with Sup cost function and threshold 0, for empty initial distributions. Reductions to threshold problems with other cost functions close our discussion to prove their EXPTIME-hardness.

Alternating Boolean Formula. We first recall the alternating Boolean formula problem (ABF) introduced as game G_6 in [15], which is the EXPTIME-hard problem from which

[1] In case of discounted-sum, we design $[\![\mathcal{G}]\!]$ with a discount factor $\sqrt{\lambda}$ (not necessarily rational), but we ensure that only one turn over two has a non-zero weight, so that we may indeed apply the reasoning of [18] and their pseudo-polynomial algorithm.

we perform our reductions. Intuitively, an ABF is an (infinite) game played on a Boolean formula whose variables are partitioned into two sets. Each player controls the values of one of the sets of variables. Players take turns changing the value of one of the variables they control. The objective of the first player (Prover) is to eventually make the formula true, while the second player (Disprover) tries to avoid this. We note that this game closely resembles an infinite horizon version of the more classical QBF Problem.

More formally, an ABF instance is given by two finite disjoint sets of Boolean variables, X and Y, and a CNF formula over $X \cup Y$. The game is played by two players called Prover and Disprover. They take turns changing the value of at most one of the variables they own (X are the variables of Prover, and Y those of Disprover). Prover wins if and only if the formula is eventually true. A configuration of this game is thus a pair $(\text{val}, \text{Player})$ where val is the current valuation of the variables and Player indicates which player should play next. The ABF problem consists in, given an ABF game and an initial configuration, determining whether Disprover has a winning strategy from the initial configuration. It is shown EXPTIME-complete in [15].

▶ **Example 3.** Consider the formula $\Phi = Cl_1 \wedge Cl_2 \wedge Cl_3 \wedge Cl_4$ where $Cl_1 = A \vee \neg C$, $Cl_2 = C \vee D$, $Cl_3 = C \vee \neg D$ and $Cl_4 = B \vee \neg B$. Let us further consider the partition of the variables into the sets $X = \{A, B\}$ of Prover, and $Y = \{C, D\}$ of Disprover; and the initial configuration $(\text{val}, \text{Prover})$, where $\text{val} = \{B, C, D\}$ (we denote a valuation by the set of all variables it sets to true). Clearly, in this initial configuration, Φ is false since Cl_1 is false. From that configuration, Prover can either set A to true, or B to false. In the former case, one obtains the configuration $(\{A, B, C, D\}, \text{Disprover})$, where Prover wins, as Φ now evaluates to true. In the latter case, one obtains the configuration $(\{C, D\}, \text{Disprover})$. We claim that, from this configuration, Prover cannot win the game anymore, i.e., Disprover has a winning strategy that consists in first setting C to false, and in, all subsequent rounds, always flipping the value of D, whatever Prover does. Playing according to this strategy ensures Disprover to force visiting only configurations where either Cl_2 or Cl_3 is false.

Extended QSG. To make the encoding of ABF instances into QSG easier, we introduce *extended quantitative sabotage games* (with Sup cost function). Those games are QSG with Sup cost function, a designated subset $F \subseteq V$ of *final vertices* and a designated subset $S \subseteq E$ of *safe edges* (those special vertices and edges are henceforth depicted with double lines). F and S influence the semantics of the game: Saboteur can place some budget on final vertices (which is accounted for in the cost when Runner visits those vertices), but cannot put budget on safe edges; and the game stops as soon as Runner visits a final vertex. We consider the *extended safety problem* (*ESPr*), which is to determine whether an extended QSG $\mathcal{G}$ *with empty initial distribution* has value $\mathbf{Val}(\mathcal{G}) \leqslant 0$.

Since the cost function is Sup, this amounts to checking that Runner has a strategy to reach a final vertex, with no budget assigned to it, without crossing any edge with non-null budget. From now on, we assume $B < |E|$, as the problem is trivial otherwise. Then:

▶ **Lemma 4.** *The ABF problem is polynomial-time reducible to ESPr.*

Proof Sketch. We consider an instance of the ABF problem given by Boolean variable sets X and Y (owned by Prover and Disprover, respectively) and a CNF formula Φ over $X \cup Y$. We construct an extended QSG $\mathcal{E}$ such that Saboteur wins in $\mathcal{E}$ if and only if Prover wins in the ABF problem. Valuations of the variables in $X \cup Y$ are encoded by budget distributions in $\mathcal{E}$. For each variable $x \in X \cup Y$, $\mathcal{E}$ has 4 *final* vertices associated with x,

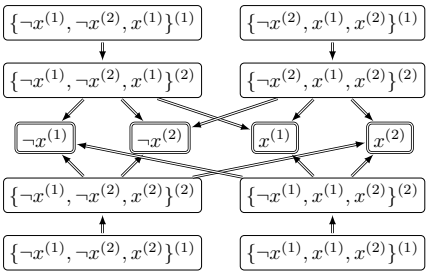


Figure 3 Verifying condition (i).

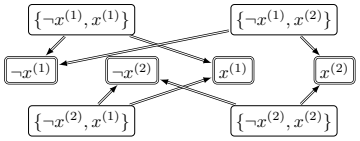


Figure 4 Verifying condition (ii).

$Ver(x) = \{\neg x^{(1)}, \neg x^{(2)}, x^{(1)}, x^{(2)}\}$. A budget distribution δ encodes a valuation in which variable $x \in X \cup Y$ is **true** if and only if $\delta(x^{(1)}) = \delta(x^{(2)}) = 1$ and $\delta(\neg x^{(1)}) = \delta(\neg x^{(2)}) = 0$.

Then, $\mathcal{E}$ simulates the ABF game as follows. The duty of Saboteur is to move the budget distribution in such a way that he respects the encoding of the valuations explained above. To enforce this, we rely on the two gadgets, depicted in Figure 3 and 4. They allow Runner to check that Saboteur respects the encoding and let him lose if he does not. More precisely, the gadget in Figure 3 allows one to check that (i) there is a non-zero budget on at least two vertices from $Ver(x)$; and the one in Figure 4 that (ii) there is a non-zero budget on exactly $\{\neg x^{(1)}, \neg x^{(2)}\}$ or $\{x^{(1)}, x^{(2)}\}$. To allow Runner to check one of these conditions, we allow him to move to one of the four corner vertices of the corresponding gadget, from where one can easily check Runner can win if and only if the condition is not respected. In our reduction, Runner will be allowed to check condition (i), for all variables, from all vertices but will be able to check (ii) only on some of them, as we will see later.

The remaining of the construction is done in a way to allow Saboteur and Runner to choose valid re-configurations of $Ver(x)$ for all variables x, and make sure that if a player cheats, it allows the other player to win the safety game. If at some point, the formula Φ becomes true, then we allow Saboteur to enter a final gadget which verifies that the current budget distribution to $Ver(X) = \bigcup_{x \in X \cup Y} Ver(x)$ satisfies Φ. This last gadget lets Runner choose a clause and then allows Saboteur to choose a literal, within this clause, which should be true. It is easy to see that the choice of clause Cl can be done by way of safe edges. The choice of literal, done by Saboteur, consists in choosing a suffix of Cl for which the left-most literal holds. Figure 5 shows the *ESPr* which results from applying our construction to the ABF formula from Example 3. ◀

We now explain how to encode safe edges and final vertices into usual QSGs, therefore showing the EXPTIME-hardness of the safety problem for QSGs.

▶ **Lemma 5.** *The extended safety problem ESPr is polynomial-time reducible to a safety problem SPr with budget* 2.

Proof Sketch. Each final vertex v in an extended QSG $\mathcal{E}$ is replaced by the gadget in Figure 6a, where $\{\alpha_i \mid 1 \leqslant i \leqslant B+1\}$ is a clique of size $B+1$, hence bigger than the budget of Saboteur. To encode $\delta(v) = 1$ in $\mathcal{E}$, Saboteur now puts one unit of budget on $(\boxed{A}, \boxed{C_1})$. If Runner reaches the gadget (through $\boxed{A}$), Saboteur puts one unit of budget on $(\boxed{A}, \boxed{C_2})$. Clearly, Runner loses if and only if there was already one unit on $(\boxed{A}, \boxed{C_1})$ (i.e., v was marked in $\mathcal{E}$). Each safe edge $(\boxed{A}, \boxed{C})$ is replaced by the gadget in Figure 6b. Here, we make use of final vertices and disjoint paths so that Saboteur cannot block all paths from $\boxed{A}$ to $\boxed{C}$ without letting Runner win by visiting a final vertex with zero budget. Both gadgets have polynomial size since we assume that $B < |E|$. ◀

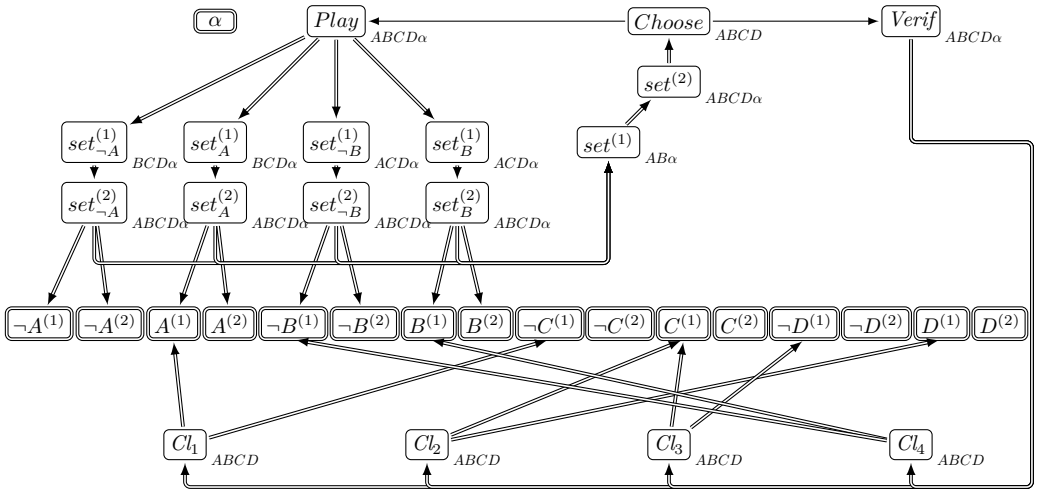


Figure 5 Excerpt of the *ESPr* constructed from the ABF of Example 3. In addition to these nodes and edges, the full *ESPr* contains: an initialisation gadget; a *safe* edge from a node n to all four corner nodes of gadget (i) in Figure 3 iff n is labeled by α; and a *safe* edge from a node n to all four corner nodes of gadget (ii) in Figure 4 testing variable $x \in \{A, B, C, D\}$ iff n is labeled by x. These parts have been omitted for the sake of clarity.

As the safety problem is a specific case of the threshold problem for Sup QSGs (where the initial distribution is empty, and threshold is fixed to 0), it follows that $ThPr_{\mathsf{Sup}}(0)$ and $ThPr_{\mathsf{Sup}}$ are EXPTIME-hard too.

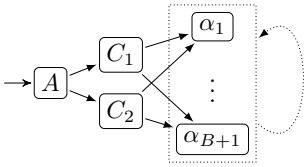
We note that given a QSG $\mathcal{G}$, for all plays π in $\mathcal{G}$, for all $0 < \lambda < 1$, and for all $\delta \in \Delta(\mathcal{G})$, $\mathsf{Sup}(\overline{\pi}) = 0$ if and only if $\mathsf{DS}_\lambda(\overline{\pi}) = 0$. This implies the following result, showing that $ThPr_{\mathsf{DS}_\lambda}(0)$ and $ThPr_{\mathsf{DS}_\lambda}$ are also EXPTIME-hard.

▶ **Lemma 6.** *For any $\lambda \in (0, 1)$, the threshold problem for DS_λ and threshold 0 is equivalent to the threshold problem for* Sup *and threshold 0.*

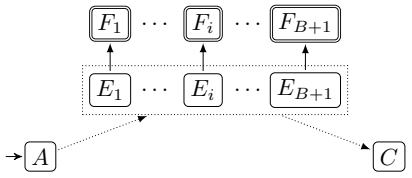
Let us now focus on LimSup. To show that $ThPr_{\mathsf{LimSup}}$ is EXPTIME-hard, we describe a reduction from *SPr* to $ThPr_{\mathsf{LimSup}}(0)$ as stated in the following lemma.

▶ **Lemma 7.** *The safety problem SPr is polynomial-time reducible to the threshold problem for* LimSup *and threshold 0.*

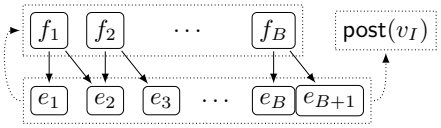
Proof Sketch. Let $\mathcal{I} = (V, E, B, v_I, \delta_I, \mathsf{Sup})$ be an instance of *SPr* (with $G(\mathcal{I})$ its underlying graph (V, E)). We build a QSG $\mathcal{G}$ with cost function LimSup such that $\mathbf{Val}(\mathcal{G}) = 0$ if and only if Runner wins in $\mathcal{I}$. The idea of the construction is that a play of $\mathcal{G}$ consists in simulating a potentially infinite sequence of plays of $\mathcal{I}$, using appropriate gadgets to 'reset' the safety game between two successive simulations. Then, repeatedly playing a winning strategy for $\mathcal{I}$ allows Runner to ensure a LimSup of 0 in $\mathcal{G}$; and one can extract a winning strategy for the safety game $\mathcal{I}$ from any strategy ensuring a LimSup of 0 in $\mathcal{G}$. The QSG $\mathcal{G}$ has budget $B' = |E|$ and is obtained by extending $G(\mathcal{I})$ with two gadgets. Note that we are giving Saboteur more budget than he had in $\mathcal{I}$. However, as we will see in the sequel, at the beginning of every faithful simulation of $\mathcal{I}$ (i.e. when Runner moves to $G(\mathcal{I})$) there will be $B' - B$ of it in the second gadget and B in the first and during any faithful simulation of $\mathcal{I}$ only budget from the initial gadget is redistribtued into $G(\mathcal{I})$.



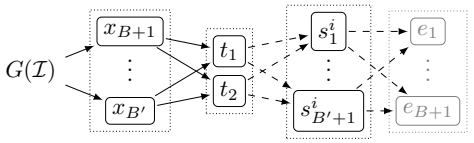
(a) A gadget for final vertices.



(b) A gadget for safe edges.



(c) Initial gadget for Sup to LimSup reduction.



(d) Exit gadget for Sup to LimSup reduction. Dashed arrows represent a (safe) path traversing B' sets s^i of vertices.

Figure 6 Dotted arrows represent edges from all sources to all targets.

The first gadget is an initial gadget which is visited every time the safety game is 'reset'. It allows Runner to stay safe from any weighted edges (and avoid reaching $G(\mathcal{I})$) until Saboteur has placed B units of budget on it (and thus removed them from the $G(\mathcal{I})$. It is depicted in Figure 6c, where all e_i are intuitively copies of v_I, and $\mathsf{post}(v_I)$ corresponds to the set of all successors of v_I in $G(\mathcal{I})$.

The second gadget allows Runner to leave $G(\mathcal{I})$ if Saboteur ever places more than B units of budget on $G(\mathcal{I})$ (and thus removes this budget from the gadgets), thereby triggering a 'reset' of the simulation. This gadget, depicted in Figure 6d, also allows Runner to come back to the initial gadget visiting only edges with zero budget. The figure shows a sequence of safe transitions (i.e. several vertices with high out-degree) which leads back to the copies e_i of the initial vertex. Further, this 'safe path' takes long enough for Saboteur to redistribute the budget from $G(\mathcal{I})$ to both gadgets. In order for Saboteur to stop Runner from always taking this 'safe exit' from $G(\mathcal{I})$ he can place $B' - B$ budget in specific edges of this second gadget. More specifically, he can place a unit of budget on one outgoing edge from each x_j, for $B + 1 \leqslant j \leqslant B'$, before forcing Runner to enter $G(\mathcal{I})$.

Intuition behind the global construction. Assume that Saboteur has a winning strategy in $\mathcal{I}$. Then, when Runner is in the initial gadget, Saboteur will play as expected and remove all weights from $G(\mathcal{I})$. Critically, the weights he removes from $G(\mathcal{I})$ will go to specific edges in both gadgets described above. Runner is now forced to play into $G(\mathcal{I})$, and Saboteur can follow his winning strategy to hit Runner at some point without using more than B weights. If Runner attempts to bail out of G through the alternative exit, and to head back to the initial gadget, then we make sure he is also hit by Saboteur. Clearly, this ensures that the LimSup value of the game is strictly greater than 0. Now assume that Runner has a winning strategy in $\mathcal{I}$. In this case, if Saboteur does not remove all weights from $G(\mathcal{I})$, then Runner is allowed to stay in the initial gadget forever or jump to $G(\mathcal{I})$ and immediately bail out using the exit gadget. In both cases he avoids getting hit by Saboteur. Let us assume Saboteur plays as expected and thus Runner enters $G(\mathcal{I})$ eventually. In this case, Runner can play his winning strategy, hence avoiding edges with non-zero budget (with Saboteur using budget B). Either he dodges weighted edges forever, or Saboteur cheats and uses some of his additional budget. However, in this case he creates an exit for Runner back to the

initial gadget, and the same analysis as above applies. This implies that the value of the game is exactly 0. ◀

Proving the EXPTIME-hardness result for cost function Avg is done by noticing that, for threshold 0, both problems are equivalent.

▶ **Lemma 8.** *The threshold problem for* LimSup *and threshold* 0 *is polynomial-time reducible to the threshold problem for* Avg *and threshold* 0*.*

4 Static quantitative sabotage games

In light of the EXPTIME-completeness of QSGs, we study in this section a restriction of the problem, that might be sufficient to model some interesting cases. The restriction concerns the dynamics of the behaviour of Saboteur. In a *static* QSG, Saboteur chooses at the beginning a budget distribution (hence, changing the initial budget distribution), and then commits to this distribution during the whole game. The situation is no longer a reactive two-player game, but rather we ask whether for every possible initial (and static) budget distribution, Runner has a nicely behaved strategy.

Formally, for a QSG $\mathcal{G} = (V, E, B, v_I, f)$ (we remove the initial budget distribution from the tuple in this section, since it is useless) and a budget distribution $\delta \in \Delta(\mathcal{G})$, we denote by $\mathcal{G}_\delta$ the QSG obtained from $\mathcal{G}$ by taking δ as initial budget distribution. Furthermore, we define the *identity strategy* ι of Saboteur in $\mathcal{G}$, as the strategy mapping every prefix $\pi \in \mathrm{Prefs}_{\mathrm{Sab}}(\mathcal{G})$ to the last budget distribution appearing in prefix π. We let $\mathbf{Val}_{\mathrm{stat}}(\mathcal{G}) = \sup_{\delta\in\Delta(\mathcal{G})} \inf_{\rho\in\Sigma_{\mathrm{Run}}(\mathcal{G})} f(\pi^\delta_{\rho,\iota})$, where $\pi^\delta_{\rho,\iota}$ denotes the unique play defined by the profile (ρ, ι) in QSG $\mathcal{G}_\delta$. Notice that this value is equal to $\inf_{\rho\in\Sigma_{\mathrm{Run}}(\mathcal{G})} \sup_{\delta\in\Delta(\mathcal{G})} f(\pi^\delta_{\rho,\iota})$, since in $\mathcal{G}$, when Saboteur follows strategy ι, the quantitative game $[\![\mathcal{G}]\!]$ is split into independent games, one for each initial distribution δ, that Runner knows as soon as it starts playing. The Static Threshold problem with cost function f consists in, given as input a QSG $\mathcal{G}$ with cost function f and a non-negative threshold T, determining whether the inequality $\mathbf{Val}_{\mathrm{stat}}(\mathcal{G}) \leqslant T$ holds. We now state the complexity of this new problem.

▶ **Theorem 9.** *For cost functions* Inf *and* LimInf*, the static threshold problem over* QSG*s is in* PTIME*; for* Sup*,* LimSup*,* Avg*, and* DS*, it is* coNP*-complete.*

First, we give the intuition behind our polynomial-time algorithm to decide the static threshold problem for cost functions Inf and LimInf.

▶ **Lemma 10.** *For cost functions* Inf *and* LimInf*, the static threshold problem over* QSG*s is in* PTIME*.*

Proof Sketch. For Inf, we claim that $\mathbf{Val}_{\mathrm{stat}}(\mathcal{G}) = \lfloor |\overline{E}|/B \rfloor$, where $\overline{E}$ is the set of edges reachable from v_I. Indeed once a distribution δ is chosen, any optimal strategy of Runner will make him reach an edge of $\overline{E}$ that has the minimum weight, thus Saboteur must distribute evenly its budget over $\overline{E}$. A similar argument works for LimInf, showing that $\mathbf{Val}_{\mathrm{stat}}(\mathcal{G}) = \lfloor |\widetilde{E}|/B \rfloor$, where $\widetilde{E}$ is the set of edges reachable from v_I and contained in a strongly connected component. ◀

Then, let us turn to the coNP-completeness of the problem for cost functions Sup, LimSup, Avg, and DS. Notice that, because of the two possible definitions of $\mathbf{Val}_{\mathrm{stat}}(\mathcal{G})$ explained in the beginning of the section, the complement of the static threshold problem asks whether there exists a budget distribution δ such that $f(\pi^\delta_{\rho,\iota}) > T$ for every strategy $\rho \in \Sigma_{\mathrm{Run}}(\mathcal{G})$

of Runner. Thus we show the NP-completeness of the complement of the static threshold problems for the four cost functions.

▶ **Lemma 11.** *For cost functions* Sup, LimSup, Avg, *and* DS, *the complement of the static threshold problem over* QSG*s is* NP*-complete.*

Proof Sketch. For the membership in NP, we can first guess a budget distribution δ (that is of size polynomial), and then compute the value of the one-player (since player Max has no choices anymore) quantitative game $\mathcal{G}_\delta$, to check if it is greater than T: computing the value of such a game can be done in polynomial time for the four cost functions we consider (see [1]).

For the NP-hardness with cost functions LimSup and Avg, we give a reduction from the following problem. The Feedback arc set problem asks, given a directed graph $G = (V, E)$ and a threshold $k \leqslant |E|$, whether there is a set E' of at most k edges of G such that $(V, E \setminus E')$ is acyclic. Karp showed [9] that the feedback arc set problem is NP-complete. Let us consider an instance of the feedback arc set problem, given by a directed graph $G = (V, E)$ and a natural integer $k \leqslant |E|$. Wlog, we can add to the graph a vertex v_I, with null in-degree, and, for all vertices $v \neq v_I$, an edge (v_I, v). Observe that this does not change the output of the feedback arc set problem as v_I is not included in any cycle. We then construct a QSG $\mathcal{G} = (V, E, k, v_I, f)$ with $f \in \{\mathsf{LimSup}, \mathsf{Avg}\}$. It is not difficult to show that $\mathbf{Val}_{\text{stat}}(\mathcal{G}) > 0$ if and only if there exists a set E' of k edges of G such that $(V, E \setminus E')$ is acyclic. The result for Sup and DS is then obtained by a slight modification of the previous proof. In particular, we make use of Lemma 6, once more. ◀

5 Reactive systems under failure

One can see a sabotage game as a system in which a controller tries to evolve while avoiding as much as possible the failures caused by the environment. The vertices of the graph represent configurations of the system, edges represent the actions, and the budget of the Saboteur may represent a finite amount of failures that can simultaneously occur during the execution. In a quantitative reasoning, a failure may be better represented by a quantity describing how much some elements of the system are overloaded, and then how much it would cost, in terms of time or energy, to use them.

Following this main motivation, we propose to look at sabotage games as a particular semantics of controllable systems. Indeed, while a standard semantics would analyse the feasibility of a requirement in a fully functional system, a *sabotage semantics* allows one to analyse systems subject to errors, and to decide, e.g., whether one can satisfy a Boolean constraint while minimising the average number of failures encountered during the execution. In particular, sabotage games, as introduced in this work, would correspond to the sabotage semantics of a system where the controller must walk in a graph with no particular objective, other than minimising the failures.

From a modelling point of view, graphs—which can be viewed as one-player games with trivial winning conditions—are quite limited. In more realistic models, we may be interested in modelling systems with uncontrollable actions (i.e., two-player games), and where the controller has a specific Boolean goal to achieve, instead of simply staying in the graph *ad vitam æternam*. A more realistic goal is usually expressed via a parity condition or LTL formulas. When a reactive system is modelled by a two-player parity game, deciding whether one can ensure the parity condition, while maintaining a cost associated with the sabotage semantics below a given threshold, can be shown to be not harder than solving sabotage

games. That is, the problem is EXPTIME-complete. This result is obtained by a reduction to quantitative parity games [4] (see [2] for a formal proof). When the requirement is expressed with an LTL formula instead of a parity condition, the problem becomes 2-EXPTIME-complete, due to an additional exponential blow-up in the size of the input formula. Note, however, that the LTL-reactive synthesis problem itself (with the standard non-sabotage semantics) is already 2-EXPTIME-complete. In this case, the sabotage semantics does not add to the complexity of the problem, which further shows that our present contributions might have practical applications, albeit the high complexity.

6 Conclusion

We have conducted a study of systems subject to failure, using the model of *quantitative sabotage games*. We have shown that under *dynamic sabotage*, the threshold problem is EXPTIME-complete for most objective functions, and coNP-complete under *static sabotage*, for the same functions (see table 1 for a summary of these results). We have also shown the applicability of our framework to deal with the more general problem of reactive synthesis in systems under failures. The QSGs we have introduced open many questions related to evolving structures. Here we have studied the worst-case scenario, i.e., where the environment is modelled by an antagonistic adversary, but, as considered in [10] for reachability Boolean objectives, one could also look at a probabilistic model, where failures, i.e., redistributions of weights, are random variables. Another natural extension of this work would be to consider a more realistic setting where the controller (Runner) has partial information regarding the weights of Saboteur.

Although the synthesis problem has been widely studied in theory, there are not many tools which implement the known theoretical solutions to decide it. The is is particularly true for quantitative objectives. Recently, however, competitions have been organised to encourage the development of such tools and the standardisation of an input format (see, e.g., SYNTCOMP and SyGuS).[2] Motivated by the similarities between the ABF problem (solving a safety game described by a logical formula) and the synthesis problem as solved in those competition (solving a safety game described by a logical circuit), one of our future projects is to show that quantitative extensions of some of the practical tools implemented for the reactive synthesis problem could be used to solve sabotage games.

References

1 Krzysztof R. Apt and Erich Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

2 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Benjamin Monmege, Guillermo A. Pérez, and Gabriel Renault. Quantitative games under failures. Research Report 1504.06744, arXiv, April 2015.

3 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.

4 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *LICS*, pages 178–187. IEEE, 2005.

5 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

[2] Links to both competitions' websites: `http://www.syntcomp.org` and `http://www.sygus.org/`.

6 Arthur S. Goldstein and Edward M. Reingold. The complexity of pursuit on a graph. *Theor. Comput. Sci.*, 143(1):93 – 112, 1995.

7 Sten Grüner, Frank G. Radmacher, and Wolfgang Thomas. Connectivity games over dynamic networks. *Theor. Comput. Sci.*, 493:46–65, 2013.

8 Marcin Jurdziński. Deciding the winner in parity games is in $\mathsf{UP} \cap \mathsf{coUP}$. *Information Processing Letters*, 68(3):119–124, 1998.

9 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

10 Dominik Klein, Frank G. Radmacher, and Wolfgang Thomas. Moving in a network under random failures: A complexity analysis. *Science of Comp. Prog.*, 77(7-8):940–954, 2012.

11 Lena Maria Kurzen. *Complexity in interaction.* PhD thesis, Institute for Logic, Language and Computation, 2011.

12 Christof Löding and Philipp Rohde. Solving the sabotage game is PSPACE-hard. In *MFCS*, volume 2747 of *LNCS*, pages 531–540. Springer, 2003.

13 Donald A Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

14 Merlijn Sevenster. *Branches of imperfect information: logic, games, and computation.* PhD thesis, Institute for Logic, Language and Computation, 2006.

15 Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM J. Comput.*, 8(2):151–174, 1979.

16 Johan van Benthem. An essay on sabotage and obstruction. In *Mechanizing Mathematical Reasoning*, volume 2605 of *LNAI*, pages 268–276. Springer, 2005.

17 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.*, 411(26-28):2433–2453, 2010.

18 Uri Zwick and Michael S. Paterson. The complexity of mean payoff games. *Theor. Comput. Sci.*, 158:343–359, 1996.

Games with Delays – A Frankenstein Approach

Dietmar Berwanger and Marie van den Bogaard

CNRS & Université Paris-Saclay, ENS Cachan, LSV, France
dwb@lsv.fr, mvdb@lsv.fr

Abstract

We investigate infinite games on finite graphs where the information flow is perturbed by non-deterministic signalling delays. It is known that such perturbations make synthesis problems virtually unsolvable, in the general case. On the classical model where signals are attached to states, tractable cases are rare and difficult to identify.

In this paper, we propose a model where signals are detached from control states, and we identify a subclass on which equilibrium outcomes can be preserved, even if signals are delivered with a delay that is finitely bounded. To offset the perturbation, our solution procedure combines responses from a collection of virtual plays following an equilibrium strategy in the instant-signalling game to synthesise, in a Dr. Frankenstein manner, an equivalent equilibrium strategy for the delayed-signalling game.

1998 ACM Subject Classification C 2.4 Distributed Systems, F.1.2 Modes of Computation

Keywords and phrases infinite games on graphs, imperfect information, delayed monitoring, distributed synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.307

1 Introduction

Appropriate behaviour of an interactive system component often depends on events generated by other components. The ideal situation, in which perfect information is available across components, occurs rarely in practice – typically a component only receives signals more or less correlated with the actual events. Apart from imperfect signals generated by the system components, there are multiple other sources of uncertainty due to actions of the system environment or to unreliable behaviour of the infrastructure connecting the components: For instance, communication channels may delay or lose signals, or deliver them in a different order than they were emitted. Coordinating components with such imperfect information to guarantee optimal system runs is a significant, but computationally challenging, problem, in particular when the interaction is of infinite duration. It appears worthwhile to study the different sources of uncertainty in separation rather than as a global phenomenon, to understand their computational impact on the synthesis of multi-component systems.

In this paper, we consider interactive systems modelled by concurrent games among multiple players with imperfect information over finite state-transition systems, or labelled graphs. Each state is associated to a stage game in which the players choose simultaneously and independently a joint action, which triggers a transition to a successor state and generates a local payoff and possibly further private signals to each player. Plays correspond to infinite paths through the graph and yield to each player a global payoff according to a given aggregation function, such as mean payoff, limit superior payoff, or parity. As solutions to such games, we are interested in synthesising Nash equilibria in pure strategies, i.e., profiles of deterministic strategies that are self-enforcing when prescribed to all players by a central coordinator.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 307–319

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The basic setting is standard for the automated verification and synthesis of reactive modules that maintain ongoing interaction with their environment seeking to satisfy a common global specification. Generally, imperfect information about the play is modelled as uncertainty about the current state in the underlying transition system, whereas uncertainty about the actions of other players is not represented explicitly. This is because the main question concerns *distributed* winning strategies, i.e., Nash equilibria in the special case where the players have a common utility function and should each receive maximal payoff. If every player wins when all follow the prescribed strategy, unilateral deviations cannot be profitable and any reaction to them would be ineffective, hence there is no need to monitor actions of other players. Accordingly, distributed winning strategies can be defined on (potential) histories of visited states, independently of the history of played actions. Nevertheless, these games are computationally intractable in general, already with respect to the question of whether distributed winning strategies exist [12, 11, 1].

However, if no equilibria exist that yield maximal payoffs to all players in a game, and we consider arbitrary Nash equilibria rather than distributed winning strategies, it becomes crucial for a player to monitor the actions of other players. To illustrate, one elementary scheme for constructing equilibria in games of infinite duration relies on *grim-trigger* strategies: cooperate on the prescribed equilibrium path until one player deviates, and at that event, enter a coalition with the remaining players and switch to a joint punishment strategy against the deviator. Most procedures for constructing Nash equilibria in games for verification and synthesis are based on this scheme, which relies essentially on the ability of players to detect *jointly* the deviation [15, 17, 5, 4].

The grim-trigger scheme works well under perfect, instant monitoring, where all players have common knowledge about the most recent action performed by any other player. In contrast, the situation becomes more complicated when players receive only imperfect signals about the actions of other players, and worse, if the signals are not delivered instantly, but with uncertain delays that may be different for each player. Imagine a scenario with three players, where Player 1 deviates from the equilibrium path and this is signalled to Player 2 immediately, but only with a delay to Player 3. If Player 2 triggers a punishment strategy against Player 1 as soon as she detects the deviation, Player 3 may monitor the action of Player 2 as a deviation from the equilibrium and trigger, in his turn, a punishment strategy against her, overthrowing the equilibrium outcome to the profit of Player 1.

Our contribution

We study the effect of imperfect, delayed monitoring on equilibria in concurrent games. Towards this, we first introduce a refined game model in which observations about actions are separated from observations about states, and we incorporate a representation for nondeterministic delays for observing action signals. To avoid the general undecidability results from the basic setting, we restrict to the case where the players have perfect information about the current state.

Our main result is that, under the assumption that the delays are uniformly bounded, every equilibrium payoff in the variant of a game where signals are delivered instantly is preserved as an equilibrium payoff in the variant where they are delayed. To prove this, we construct strategies for the delayed-monitoring game by combining responses for the instant-monitoring variant in such a way that any play with delayed signals corresponds to a shuffle of several plays with instant signals, which we call threads. Intuitively, delayed-monitoring strategies are constructed, in a Frankenstein manner, from a collection of instant-monitoring equilibrium strategies. Under an additional assumption that the payoff structure is insensitive

to shuffling plays, this procedure allows to transfer equilibrium payoffs from the instant to the delayed-monitoring game.

Firstly, the transfer result can be regarded as an equilibrium existence theorem for games with delayed monitoring based on classes of games with instant monitoring that admit equilibria in pure strategies. Defining existence conditions is a fundamental prerequisite to using Nash equilibrium as a solution concept. If an application model leads to games that may not admit equilibria, this is a strong reason to look for another solution concept. As mixed strategies are conceptually challenging in the context of infinite games, guarantees for pure equilibrium existence are particularly desirable.

Secondly, our result establishes an outcome equivalence between games with instant and delayed monitoring, within the given restrictions: As the preservation of equilibrium values from delayed-monitoring games to the instant-monitoring variant holds trivially (the players may just buffer the received signals until an admissible delay period passed, and then respond), we obtain that the set of pure equilibrium payoffs is the same, whether signals are delayed or not — although, of course, the underlying equilibrium strategies differ between the two variants. In terms of possible equilibrium payoffs, these games are hence robust under changing signalling delivery guarantees, as long as the maximal delays are commonly known. In particular, payoff-related results obtained for the instant-signalling variant apply directly to the delayed variant.

Thirdly, the transfer procedure has some algorithmic content. When we set out with finite-state equilibrium strategies for the instant-monitoring game, the procedure will also yield a profile of finite-state strategies for the delayed-monitoring game. Hence, the construction is effective, and can be readily applied to cases where synthesis procedures for finite-state equilibria in games with instant monitoring exist.

Related literature

One motivation for studying infinite games with delays comes from the work of Shmaya [14] considering sequential games on finitely branching trees (or equivalently, on finite graphs) where the actions of players are monitored perfectly, but with arbitrary finite delays. In the setting of two-player zero-sum games with Borel winning conditions, Shmaya shows that these delayed-monitoring games are determined in mixed strategies. Apart of revealing that infinite games on finite graphs are robust under monitoring delays, the paper is enlightening for its proof technique which relies on a reduction of the delayed-monitoring game to a game with a different structure that features instant monitoring but, in exchange, involves stochastic moves.

Our analysis is inspired directly from recent work of Fudenberg, Ishii, and Kominers [8] on infinitely repeated games with bounded-delay monitoring whith stochastically distributed observation lags. The authors prove a transfer result that is much stronger than ours, which also covers the relevant case of discounted payoffs (modulo a controlled adjustment of the discount factor). The key idea for constructing strategies in the delayed-response game is to modify strategies from the instant-response game by letting them respond with a delay equal to the maximal monitoring delay so that all players received their signals. This amounts to combining different threads of the instant-monitoring game, one for every time unit in the delay period. Thus, the proof again involves a reduction between games of different structure, with the difference that here one game is reduced to several instances of another one.

Infinitely repeated games correspond to the particular case of concurrent games with only one state. This allows applying classical methods from strategic games which are no longer accessible in games with several states [13]. Additionally, the state-transition structure of our

setting induces a combinatorial effort to adapt the delayed-response strategies from [8]: As the play may reach a different state until the monitoring delay expires, the instant-monitoring threads must be scheduled more carefully to make sure that they combine to a valid play of the delayed-monitoring variant. In particular, the time for returning to a particular game state may be unbounded, which makes it hard to deliver guarantees under discounted payoff functions. As a weaker notion of patience, suited for games with state transitions, we consider payoff aggregation functions that are *shift-invariant and submixing*, as introduced by Gimbert and Kelmendi in their work on memoryless strategies in stochastic games [9].

Our model generalises concurrent games of infinite duration over finite graphs. Equilibria in such models have been investigated for the perfect-information case, and it was shown that it is decidable with relatively low complexity whether equilibria exist, and if this is the case, finite-state equilibrium profiles can be synthesised for several cases of interest in automated verification. Ummels [15] considers turned-based games with parity conditions and shows that deciding whether there exists a pure Nash equilibrium payoff in a given range is an NP-complete problem. For the case of concurrent games with mean-payoff conditions, Ummels and Wojtczak [16], show that the problem for pure strategies is still NP-complete, whereas it becomes undecidable for mixed strategies. For the case of concurrent games with Büchi conditions, that is, parity conditions with priorities 1 and 2, Bouyer et al. [3] show that the complexity of the problem drops to PTime. These results are in the setting of perfect information about the actual game state and perfect monitoring. However, as pointed out in the conclusion of [3], the generic complexity increases when actions are not monitored by any player.

The basic method for constructing equilibria in the settings of perfect monitoring relies on grim-trigger strategies that react to deviations from the equilibrium path by turning to a zero-sum coalition strategy opposing the deviating player. Such an approach can hardly work under imperfect monitoring where deviating actions cannot be observed directly. Alternative approaches for constructing equilibria without relying on perfect monitoring comprise, on the one hand distributed winning strategies for games that allow all players of a coalition to attain the most efficient outcome [10, 7, 2], and at the other extreme, Doomsday equilibria, proposed by Chatterjee et al. in [6], for games where any deviation leads to the most inefficient outcome, for all players.

2 Games with delayed signals

There are n players $1, \dots, n$ and a distinguished agent called Nature. We refer to a list $x = (x^i)_{1 \le i \le n}$ that associates one element x^i to every player i as a *profile*. For any such profile, we write x^{-i} to denote the list $(x^j)_{1 \le j \le n, j \neq i}$ where the element of Player i is omitted. Given an element x^i and a list x^{-i}, we denote by (x^i, x^{-i}) the full profile $(x^i)_{1 \le i \le n}$. For clarity, we always use superscripts to specify to which player an element belongs. If not quantified explicitly, we refer to Player i to mean any arbitrary player.

2.1 General model

For every player i, we fix a set A^i of *actions*, and a set Y^i of *signals*; these sets are finite. The *action space* A consists of all action profiles, and the *signal space* Y of all signal profiles.

2.1.1 Transition structure

The transition structure of a game is described by a *game graph* $G = (V, E)$ over a finite set V of *states* with an edge relation $E \subseteq V \times A \times Y \times V$ that represents *transitions* labelled by action and signal profiles. We assume that for each state v and every action profile a, there exists at least one transition $(v, a, y, v') \in E$.

The game is played in stages over infinitely many periods starting from a designated initial state $v_0 \in V$ known to all players. In each period $t \geq 1$, starting in a state v_{t-1}, every player i chooses an action a_t^i, and Nature chooses a transition $(v_{t-1}, a_t, y_t, v_t) \in E$, which determines a profile y_t of emitted signals and a successor state v_t. Then, each player i observes a set of signals depending on the monitoring structure of the game, and the play proceeds to period $t + 1$ with v_t as the new state.

Accordingly, a *play* is an infinite sequence $v_0,\ a_1, y_1, v_1,\ a_2, y_2, v_2\ \cdots \in V(AYV)^\omega$ such that $(v_{t-1}, a_t, y_t, v_t) \in E$, for all $t \geq 1$. A *history* is a finite prefix $v_0,\ a_1, y_1, v_1,\ \ldots,\ a_t, y_t, v_t \in V(AYV)^*$ of a play. We refer to the number of stages played up to period t as the *length* of the history.

2.1.2 Monitoring structure

We assume that each player i always knows the current state v and the action a^i she is playing. However, she is not informed about the actions or signals of the other players. Furthermore, she may observe the signal y_t^i emitted in a period t only in some later period or, possibly, never at all.

The signals observed by Player i are described by an *observation* function

$$\gamma^i : V(AYV)^+ \to 2^{Y^i},$$

which assigns to every nontrivial history $\pi = v_0,\ a_1, y_1, v_1,\ \ldots,\ a_t, y_t, v_t$ with $t \geq 1$, a set of signals that were actually emitted along π for the player:

$$\gamma^i(\pi) \subseteq \{y_r^i \in Y^i \mid 1 \leq r \leq t\}.$$

For an actual history $\pi \in V(AYV)^*$, the *observed history* of Player i is the sequence

$$\beta^i(\pi) := v_0,\ a_1^i, z_1^i, v_1,\ \ldots,\ a_t^i, z_t^i, v_t$$

with $z_r^i = \gamma^i(v_0,\ a_1, y_1, v_1,\ \ldots,\ a_r, y_r, v_r)$, for all $1 \leq r \leq t$. Analogously, we define the *observed play* of Player i.

A *strategy* for player i is a mapping $s^i : V(A^i 2^{Y^i} V)^* \to A^i$ that associates to every observation history $\pi \in V(A^i 2^{Y^i} V)^*$ an action $s^i(\pi)$. The *strategy space* S is the set of all strategy profiles. We say that a history or a play π *follows* a strategy s^i, if $a_{t+1}^i = s^i(\beta^i(\pi_t))$, for all histories π_t of length $t \geq 0$ in π. Likewise, a history or play follows a profile $s \in S$, if it follows the strategy s^i of each player i. The *outcome* $\mathrm{out}(s)$ of a strategy profile s is the set of all plays that follow it. Note that the outcome of a strategy profile generally consist of multiple plays, due to the different choices of Nature.

Strategies may be partial functions. However, we require that for any history π that follows a strategy s^i, the observed history $\beta^i(\pi)$ is also included in the domain of s^i.

With the above definition of a strategy, we implicitly assume that players have perfect recall, that is, they may record all the information acquired along a play. Nevertheless, in certain cases, we can restrict our attention to strategy functions computable by automata with finite memory. In this case, we speak of *finite-state strategies*.

2.1.3 Payoff structure

Every transition taken in a play generates an integer payoff to each player i, described by a *payoff* function $p^i : E \to \mathbb{Z}$. These stage payoffs are combined by a *payoff aggregation* function $u : \mathbb{Z}^\omega \to \mathbb{R}$ to determine the *utility* received by Player i in a play π as $u^i(\pi) := u(\, p^i(v_0, a_1, y_1, v_1), p^i(v_1, a_2, y_2, v_2), \dots)$. Thus, the profile of *utility*, or global payoff, functions $u^i : V(AYV)^\omega \to \mathbb{R}$ is represented by a profile of payoff functions p^i and an aggregation function u, which is common to all players.

We generally consider utilities that depend only on the observed play, that is, $u^i(\pi) = u^i(\pi')$, for any plays π, π' that are indistinguishable to Player i, that is, $\beta^i(\pi) = \beta^i(\pi')$. To extend payoff functions from plays to strategy profiles, we set

$$u^i(s) := \inf\{u^i(\pi) \mid \pi \in \mathrm{out}(s)\}, \text{ for each strategy profile } s \in S.$$

Overall, a game $\mathcal{G} = (G, \gamma, u)$ is described by a game graph with a profile of observation functions and one of payoff functions. We are interested in *Nash equilibria*, that is, strategy profiles $s \in S$ such that $u^i(s) \geq u^i(r^i, s^{-i})$, for every player i and every strategy $r^i \in S^i$. The payoff $w = u^i(s)$ generated by an equilibrium $s \in S$ is called an equilibrium payoff. An equilibrium payoff w is *ergodic* if it does not depend on the initial state of the game, that is, there exists a strategy profile s with $u(s) = w$, for every choice of an initial state.

2.2 Instant and bounded-delay monitoring

We focus on two particular monitoring structures, one where the players observe their component of the signal profile instantly, and one where each player i observes his private signal emitted in period t in some period $t + d_t^i$, with a bounded delay $d_t^i \in \mathbb{N}$ chosen by Nature.

Formally, a game with *instant monitoring* is one where the observation functions γ^i return, for every history $\pi = v_0,\ a_1, y_1, v_1,\ \dots,\ a_t, y_t, v_t$ of length $t \geq 1$, the private signal emitted for Player i in the current stage, that is, $\gamma^i(\pi) = \{y_t^i\}$, for all $t \geq 1$. As the value is always a singleton, we may leave out the enclosing set brackets and write $\gamma^i(\pi) = y_t^i$.

To model bounded delays, we consider signals with an additional component that represents a timestamp. Concretely, we fix a set B^i of *basic signals* and a finite set $D^i \subseteq \mathbb{N}$ of *possible delays*, for each player i, and consider the product $Y^i := B^i \times D^i$ as a new set of signals. Then, a game with *delayed monitoring* is a game over the signal space Y with observation functions γ^i that return, for every history $\pi = v_0,\ a_1, (b_1, d_1), v_1,\ \dots,\ a_t, (b_t, d_t), v_t$ of length $t \geq 1$, the value

$$\gamma^i(\pi) = \{(b_r^i, d_r^i) \in B^i \times D^i \mid r \geq 1,\ r + d_r^i = t\}.$$

In our model, the role of Nature is limited to choosing the delays for observing the emitted signals. Concretely, we postulate that the basic signals and the stage payoffs associated to transitions are determined by the current state and the action profile chosen by the players, that is, for every global state v and action profile a, there exists a unique profile b of basic signals and a unique state v' such that $(v, a, (b, d), v') \in E$, for some $d \in D$; moreover, for any other delay profile $d' \in D$, we require $(v, a, (b, d'), v') \in E$, and also that $p^i(v, a, (b, d), v') = p^i(v, a, (b, d'), v')$. Here again, D denotes the *delay space* composed of the sets D^i. Notice that under this assumption, the plays in the outcome of a strategy profile s differ only by the value of the delays. In particular, all plays in $\mathrm{out}(s)$ yield the same payoff.

To investigate the effect of observation delays, we will relate the delayed and instant-monitoring variants of a game. Given a game $\mathcal{G}$ with delayed monitoring, the corresponding

instant-monitoring game $\mathcal{G}'$ is obtained by projecting every signal $y^i = (b^i, d^i)$ onto its first component b^i and then taking the transition and payoff structure induced by this projection. As we assume that transitions and payoffs are independent of delays, the operation is well defined.

Conversely, given a game $\mathcal{G}$ with instant monitoring and a delay space D, the corresponding game $\mathcal{G}'$ with delayed monitoring is obtained by extending the set B^i of basic signals in $\mathcal{G}$ to $B^i \times D^i$, for each player i, and by lifting the transition and payoff structure accordingly. Thus, the game $\mathcal{G}'$ has the same states as $\mathcal{G}$ with transitions $E' := \{(v, a, (b, d), w) \mid (v, a, b, w) \in E, d \in D\}$, whereas the payoff functions are given by ${p'}^i(v, a, (b, d), w) := p^i(v, a, b, w)$, for all $d \in D$.

As the monitoring structure of games with instant or delayed monitoring is fixed, it is sufficient to describe the game graph together with the profile of payoff functions, and to indicate the payoff aggregation function. It will be convenient to include the payoff associated to a transition as an additional edge label and thus represent the game simply as a pair $\mathcal{G} = (G, u)$ consisting of a finite labelled game graph and an aggregation function $u : \mathbb{Z}^\omega \to \mathbb{R}$.

2.3 Shift-invariant, submixing utilities

Our result applies to a class of games where the payoff-aggregation functions are invariant under removal of prefix histories and shuffling of plays. Gimbert and Kelmendi [9] identify these properties as a guarantee for the existence of simple strategies in stochastic zero-sum games.

A function $f : \mathbb{Z}^\omega \to \mathbb{R}$ is *shift-invariant*, if its value does not change when adding an arbitrary finite prefix to the argument, that is, for every sequence $\alpha \in \mathbb{Z}^\omega$ and each element $a \in \mathbb{Z}$, we have $f(a\alpha) = f(\alpha)$.

An infinite sequence $\alpha \in \mathbb{Z}^\omega$ is a *shuffle* of two sequences $\varphi, \eta \in \mathbb{Z}^\omega$, if $\mathbb{N}$ can be partitioned into two infinite sets $I = \{i_0, i_1, \dots\}$ and $J = \{j_0, j_1, \dots\}$ such that $\alpha_{i_k} = \varphi_k$ and $\alpha_{j_k} = \eta_k$, for all $k \in \mathbb{N}$. A function $f : \mathbb{Z}^\omega \to \mathbb{R}$ is called *submixing* if, for every shuffle α of two sequences $\varphi, \eta \in \mathbb{Z}^\omega$, we have

$$\min\{f(\varphi), f(\eta)\} \leq f(\alpha) \leq \max\{f(\varphi), f(\eta)\}.$$

In other words, the image of a shuffle product always lies between the images of its factors.

The proof of our theorem relies on payoff aggregation functions $u : \mathbb{Z}^\omega \to \mathbb{R}$ that are shift-invariant and submixing. Many relevant game models used in economics, game theory, and computer science satisfy this restriction. Prominent examples are mean payoff or limsup payoff, which aggregate sequences of stage payoffs $p_1, p_2, \dots \in \mathbb{Z}^\omega$ by setting:

$$\text{mean-payoff}(p_1, p_2, \dots) := \limsup_{t \geq 1} \frac{1}{t} \sum_{r=1}^{t} p_r, \quad \text{and}$$

$$\text{limsup}(p_1, p_2, \dots) := \limsup_{t \geq 1} p_t.$$

Finally, parity conditions which map non-negative integer payoffs $p_1, p_2, \dots$ called priorities to $\text{parity}(p_1, p_2, \dots) = 1$ if the least priority that occurs infinitely often is even, and 0 otherwise, also satisfy the conditions.

2.4 The transfer theorem

We are now ready to formulate our result stating that, under certain restrictions, equilibrium profiles from games with instant monitoring can be transferred to games with delayed monitoring.

▶ **Theorem 2.1.** *Let $\mathcal{G}$ be a game with instant monitoring and shift-invariant submixing payoffs, and let D be a finite delay space. Then, for every ergodic equilibrium payoff w in $\mathcal{G}$, there exists an equilibrium of the D-delayed monitoring game $\mathcal{G}'$ with the same payoff w.*

The proof relies on constructing a strategy for the delayed-monitoring game while maintaining a collection of virtual plays of the instant-monitoring game on which the given strategy is queried. The responses are then combined according to a specific schedule to ensure that the actual play arises as a shuffle of the virtual plays.

3 Proof

Consider a game $\mathcal{G} = (G, u)$ with instant monitoring where the payoff aggregation function u is shift-invariant and submixing, and suppose that $\mathcal{G}$ admits an equilibrium profile s. For an arbitrary finite delay space D, let $\mathcal{G}'$ be the delayed-monitoring variant of $\mathcal{G}$. In the following steps, we will construct a strategy profile s' for $\mathcal{G}'$, that is in equilibrium and yields the same payoff $u(s')$ as s in $\mathcal{G}$.

3.1 Unravelling small cycles

To minimise the combinatorial overhead for scheduling delayed responses, it is convenient to ensure that, whenever the play returns to a state v, the signals emitted at the previous visit at v have been received by all players. If every cycle in the given game graph G is at least as long as any possible delay, this is clearly satisfied. Otherwise, the graph can be expanded to avoid small cycles, e.g., by taking the product with a cyclic group of order equal to the maximal delay.

Concretely, let m be the greatest delay among $\max D^i$, for all players i. We define a new game graph $\hat{G}$ as the product of G with the additive group $\mathbb{Z}_m$ of integers modulo m, over the state set $\{v_j \mid v \in V, j \in \mathbb{Z}_m\}$ by allowing transitions (v_j, a, b, v'_{j+1}), for every $(v, a, b, v') \in E$ and all $j \in \mathbb{Z}_m$, and by assigning stage payoffs $\hat{p}^i(v_j, a, b, v'_{j+1}) := p^i(v, a, b, v')$, for all transitions $(v, a, b, v') \in E$. Obviously, every cycle in this game has length at least m. Moreover, the games $(\hat{G}, u)$ and (G, u) are equivalent: Since the index component $j \in \mathbb{Z}_m$ is not observable to the players, the two games have the same sets of strategies, and profiles of corresponding strategies yield the same observable play outcome, and hence the same payoffs.

In conclusion, we can assume without loss of generality that each cycle in the game graph G is longer than the maximal delay $\max D^i$, for all players i.

3.2 The Frankenstein procedure

We describe a strategy f^i for Player i in the delayed monitoring game G' by a reactive procedure that receives observations of states and signals as input and produces actions as output.

The procedure maintains a collection of virtual plays of the instant-monitoring game. More precisely, these are observation histories for Player i following the strategy s^i in G, which we call *threads*. The observations collected in a thread $\pi = v_0,\ a^i_1, (b^i_1, d^i_1), v_1,\ \ldots,\ a^i_r, (b^i_r, d^i_r), v_r$ are drawn from the play of the main delayed-monitoring game G'. Due to delays, it may occur that the signal (b^i_r, d^i_r) emitted in the last period of a thread has not yet been received. In this case, the signal entry is replaced by a special symbol #, and we say that the thread is *pending*. As soon as the player receives the signal, the placeholder # is overwritten with the actual value, and the thread becomes *active*. Active threads π are used to query the

strategy s^i; the prescribed action $a^i = s^i(\pi)$ is played in the main delayed-monitoring game and it is also used to continue the thread of the virtual instant-monitoring game.

To be continued, a thread must be active and its current state needs to match the actual state of the play in the delayed-monitoring game. Intuitively, threads advance more slowly than the actual play, so we need multiple threads to keep pace with it. Here, we use a collection of $|V| + 1$ threads, indexed by an ordered set $K = V \cup \{\varepsilon\}$. The main task of the procedure is to schedule the continuation of threads. To do so, it maintains a data structure (τ, h) that consists of the threads $\tau = (\tau_k)_{k \in K}$ and a scheduling sequence $h = h[0], \ldots, h[t]$ of indices from K, at every period $t \geq 0$ of the actual play. For each previous $r < t$, the entry $h[r]$ points to the thread according to which the action of period $r + 1$ in the actual play has been prescribed; the last entry $h[t]$ points to an active thread that is currently scheduled for prescribing the action to be played next.

The version of Procedure Frankenstein^i for Player i, given below, is parametrised by the game graph G with the designated initial state, the delay space D^i, and the given equilibrium strategy s^i in the instant-monitoring game. In the initialisation phase, the initial state v_0 is stored in the initial thread τ_ε to which the current scheduling entry $h[0]$ points. The remaining threads are initialised, each with a different position from V. Then, the procedure enters a non-terminating loop along the periods of the actual play. In every period t, it outputs the action prescribed by strategy s^i for the current thread scheduled by $h[t]$ (Line 5). Upon receiving the new state, this current thread is updated by recording the played action and the successor state; as the signal emitted in the instant-monitoring play is not available in the delayed-monitoring variant, it is temporarility replaced by #, which marks the current thread as pending (Line 7). Next, an active thread that matches the new state is scheduled (Line 9), and the received signals are recorded with the pending threads to which they belong (Line 11 – 14). As a consequence, these threads become active.

3.3 Correctness

In the following, we argue that the procedure Frankenstein^i never violates the assertions in Line 4, 8, and 13 while interacting with Nature in the delayed-monitoring game $\mathcal{G}'$, and thus implements a valid strategy for Player i.

Specifically, we show that for every history

$$\pi = v_0,\ a_1, (b_1, d_1), v_1,\ \ldots,\ a_t, (b_t, d_t), v_t$$

in the delayed-monitoring game that follows the prescriptions of the procedure up to period $t > 0$, (1) the scheduling function $h[t] = k$ points to an active thread τ_k that ends at state v_t, and (2) for the state v_{t+1} reached by playing $a_{t+1} := s^i(\tau_k)$ at π, there exists an active thread $\tau_{k'}$ that ends at v_{t+1}. We proceed by induction over the period t. In the base case, both properties hold, due to the way in which the data structure is initialised: the (trivial) thread τ_ε is active, and for any successor state v_1 reached by $a_1 := s^i(\tau_\varepsilon)$, there is a fresh thread τ_{v_1} that is active. For the induction step in period $t + 1$, property (1) follows from property (2) of period t. To verify that property (2) holds, we distinguish two cases. If v_{t+1} did not occur previously in π, the initial thread $\tau_{v_{t+1}}$ still consists of the trivial history v_{t+1}, and it is thus active. Else, let $r < t$ be the period in which v_{t+1} occurred last. Then, for $k' = h[r]$, the thread $\tau_{k'}$ ends at v_{t+1}. Moreover, by our assumption that the cycles in G are longer than any possible delay, it follows that the signals emitted in period $r < t - m$ have been received along π and were recorded (Line 12–14). Hence, $\tau_{k'}$ is an active thread ending at v_{t+1}, as required.

Procedure: Frankenstein$^i(G, v_0, D^i, s^i)$

```
// initialisation
1  τ_ε := v_0; h[0] = ε
2  foreach v ∈ V do τ_v := v

// play loop
for t = 0 to ω do
3    k := h[t]
4    assert (τ_k is an active thread)
5    play action a^i := s^i(τ_k)                    // a^i_{t+1}
6    receive new state v                            // v_{t+1}
7    update τ_k := τ_k a^i # v
8    assert (there exists an index k' ≠ k such that τ_{k'} ends at state v)
9    set h[t+1] to the least such index k'
10   receive observation z^i ⊆ B^i × D^i             // z^i_{t+1}
11   foreach (b^i, d^i) ∈ z^i do
12     k := h[t − d^i]
13     assert (τ_k = ρ#v', for some prefix ρ, state v')
14     update τ_k := ρ(b^i, d^i)v'
     end
end
```

To see that the assertion of Line 13 is never violated, we note that every observation history $\beta^i(\pi)$ of the actual play π in $\mathcal{G}'$ up to period t corresponds to a finitary shuffle of the threads τ in the t-th iteration of the play loop, described by the scheduling function h: The observations $(a^i_r, (b_r, d_r)^i, v_r)$ associated to any period $r \leq t$ appear at the end of $\tau_{h[r]}$, if the signal $(b_r, d_r)^i$ was delivered until period t, and with the placeholder #, otherwise.

In summary, it follows that the reactive procedure Frankenstein^i never halts, and it returns an action for every observed history $\beta^i(\pi)$ associated to an actual history π that follows it. Thus, the procedure defines a strategy $f^i : V(A^i 2^{Y^i} V)^* \to A^i$ for Player i.

3.4 Equilibrium condition

Finally, we show that the interplay of the strategies f^i described by the reactive procedure Frankenstein^i, for each player i, constitutes an equilibrium profile for the delayed-monitoring game $\mathcal{G}'$ yielding the same payoff as s in $\mathcal{G}$.

According to our remark in the previous subsection, every transition taken in a play π that follows the strategy f^i in $\mathcal{G}'$ is also observed in some thread history, which in turn follows s^i. Along the non-terminating execution of the reactive Frankenstein^i procedure, some threads must be scheduled infinitely often, and thus correspond to observations of plays in the perfect-monitoring game G. We argue that the observation by Player i of a play that follows the strategy f^i corresponds to a shuffle of such infinite threads (after discarding finite prefixes).

To make this more precise, let us fix a play π that follows f^i in $\mathcal{G}'$, and consider the infinite scheduling sequence $h[0], h[1], \ldots$ generated by the procedure. Since there are finitely

many thread indices, some must appear infinitely often in this sequence; we denote by $L^i \subseteq K$ the subset of these indices, and look at the least period ℓ^i, after which only threads in L^i are scheduled. Then, the suffix of the observation $\beta^i(\pi)$ from period ℓ^i onwards can be written as a $|L^i|$-partite shuffle of suffixes of the threads τ_k for $k \in L^i$.

By our assumption that the payoff aggregation function u is shift-invariant and submixing, it follows that the payoff $u^i(\pi)$ lies between $\min\{u^i(\tau_k) \mid k \in L^i\}$ and $\max\{u^i(\tau_k) \mid k \in L^i\}$. Now, we apply this reasoning to all players to show that f^i is an equilibrium profile with payoff $u(s)$.

To see that the profile f in the delayed-monitoring game $\mathcal{G}'$ yields the same payoff as s in the instant-monitoring game $\mathcal{G}$, consider the unique play π that follows f, and construct L^i, for all players i, as above. Then, all threads of all players i follow s^i, which by ergodicity implies, for each infinite thread τ_k with $k \in L^i$ that $u^i(\tau_k) = u^i(s)$. Hence $\min\{u^i(\tau_k) \mid k \in L^i\} = \max\{u^i(\tau_k) \mid k \in L^i\} = u^i(\pi)$, for each player i, and therefore $u(f) = u(s)$.

To verify that f is indeed an equilibrium profile, consider a strategy g^i for the delayed-monitoring game and look at the unique play π that follows (f^{-i}, g^i) in $\mathcal{G}'$. Towards a contradiction, assume that $u^i(\pi) > u^i(f)$. Since $u^i(\pi) < \max\{u^i(\tau_k) \mid k \in L^i\}$, there must exist an infinite thread τ_k with index $k \in L^i$ such that $u^i(\tau_k) > u^i(f) = u^i(s)$. But τ_k corresponds to the observation $\beta^i(\rho)$ of a play ρ that follows s^{-i} in $\mathcal{G}$, and since s is an equilibrium strategy we obtain $u^i(s) \geq u^i(\rho) = u^i(\tau_k)$, a contradiction. This concludes the proof of our theorem.

3.5 Finite-state strategies

The transfer theorem makes no assumption on the complexity of equilibrium strategies in the instant-monitoring game at the outset; informally, we may think of these strategies as oracles that the Frankenstein procedure can query. Moreover, the procedure itself runs for infinite time along the periods the play, and the data structure it maintains grows unboundedly.

However, if we set out with an equilibrium profile of finite-state strategies, it is straightforward to rewrite the Frankenstein procedure as a finite-state automaton: instead of storing the full histories of threads, it is sufficient to maintain the current state reached by the strategy automaton for the relevant player after reading this history, over a period that is sufficiently long to cover all possible delays.

▶ **Corollary 3.1.** *Let $\mathcal{G}$ be a game with instant monitoring and shift-invariant submixing payoffs, and let D be a finite delay space. Then, for every ergodic payoff w in $\mathcal{G}$ generated by a profile of finite-state strategies, there exists an equilibrium of the D-delayed monitoring game $\mathcal{G}'$ with the same payoff w that is also generated by a profile of finite-state strategies.*

4 Conclusion

We presented a transfer result that implies effective solvability of concurrent games with a particular kind of imperfect information, due to imperfect monitoring of actions, and delayed delivery of signals. This is a setting where we cannot rely on grim-trigger strategies, typically used for constructing Nash equilibria in games of infinite duration for automated verification. Our method overcomes this obstacle by adapting the idea of delayed-response strategies of [8] from infinitely repeated games, with one state, to arbitrary finite state-transition structures.

Our transfer result imposes stronger restrictions than the one in [8], in particular, it does not cover discounted payoff functions. Nevertheless, the class of submixing payoff functions is general enough to cover most applications relevant in automated verification and synthesis.

The restriction to ergodic payoffs was made for technical convenience. We believe it is not critical for using the main result: The state space of every game can be partitioned into ergodic regions, where all initial states lead to the same equilibrium value. As the outcome of every equilibrium profile will stay within an ergodic region, we may analyse each ergodic region in separation, and apply standard zero-sum techniques to combine the results. A challenging open question is whether the assumption of perfect information about the current state can be relaxed.

Acknowledgement. This work was supported by the Indo-French Formal Methods Lab (LIA Informel) and by the European Union Seventh Framework Programme under Grant Agreement 601148 (CASSTING).

References

1 Salman Azhar, Gary Peterson, and John Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *Journal of Computers and Mathematics with Applications*, 41:957–992, 2001.

2 Dietmar Berwanger, Łukasz Kaiser, and Bernd Puchala. Perfect-information construction for coordination in games. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011), Proc.*, volume 13 of *Leibniz International Proceedings in Informatics*, pages 387–398, Mumbai, India, December 2011. Leibniz-Zentrum für Informatik.

3 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Nash equilibria in concurrent games with Büchi objectives. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011), Proc.*, volume 13 of *LIPIcs*, pages 375–386. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

4 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent games. *Logical Methods in Computer Science*, 11(2:9), June 2015.

5 Thomas Brihaye, Véronique Bruyére, and Julie De Pril. On equilibria in quantitative games with reachability/safety objectives. *Theory of Computing Systems*, 54(2):150–189, 2014.

6 Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. In KennethL. McMillan and Xavier Rival, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 8318 of *Lecture Notes in Computer Science*, pages 78–97. Springer Berlin Heidelberg, 2014.

7 B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Logic in Computer Science (LICS'05), Proc.*, pages 321–330. IEEE, 2005.

8 Drew Fudenberg, Yuhta Ishii, and Scott Duke Kominers. Delayed-response strategies in repeated games with observation lags. *J. Economic Theory*, 150:487–514, 2014.

9 Hugo Gimbert and Edon Kelmendi. Two-player perfect-information shift-invariant submixing stochastic games are half-positional. *CoRR*, abs/1401.6575, 2014.

10 Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *Logic in Computer Science (LICS'01), Proc.*, pages 389–398. IEEE Computer Society Press, June 2001.

11 Gary L. Peterson and John H. Reif. Multiple-Person Alternation. In *Proc 20th Annual Symposium on Foundations of Computer Science, (FOCS 1979)*, pages 348–363. IEEE, 1979.

12 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science, (FoCS '90)*, pages 746–757. IEEE Computer Society Press, 1990.

13 Dinah Rosenberg, Eilon Solan, and Nicolas Vieille. Stochastic games with imperfect monitoring. In Alain Haurie, Shigeo Muto, LeonA. Petrosjan, and T.E.S. Raghavan, editors, *Advances in Dynamic Games*, volume 8 of *Annals of the International Society of Dynamic Games*, pages 3–22. Birkhäuser Boston, 2006.

14 Eran Shmaya. The determinacy of infinite games with eventual perfect monitoring. *Proc. Am. Math. Soc.*, 139(10):3665–3678, 2011.

15 Michael Ummels. The complexity of Nash equilibria in infinite multiplayer games. In Roberto Amadio, editor, *Foundations of Software Science and Computational Structures*, volume 4962 of *Lecture Notes in Computer Science*, pages 20–34. Springer Berlin Heidelberg, 2008.

16 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in limit-average games. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory*, volume 6901 of *Lecture Notes in Computer Science*, pages 482–496. Springer Berlin Heidelberg, 2011.

17 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in stochastic multiplayer games. *Logical Methods in Computer Science*, 7(3), 2011.

Forbidden Extension Queries

Sudip Biswas[1], Arnab Ganguly[1], Rahul Shah[1,2], and Sharma V. Thankachan[3]

1 **Louisiana State University, Baton Rouge, USA**
{sbiswa7,agangu4}@lsu.edu,rahul@csc.lsu.edu
2 **National Science Foundation, USA**
rshah@nsf.gov
3 **Georgia Institute of Technology, Atlanta, USA**
sharma.thankachan@gatech.edu

Abstract

Document retrieval is one of the most fundamental problem in information retrieval. The objective is to retrieve all documents from a document collection that are relevant to an input pattern. Several variations of this problem such as ranked document retrieval, document listing with two patterns and forbidden patterns have been studied. We introduce the problem of document retrieval with forbidden extensions.

Let $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ be a collection of D string documents of n characters in total, and P^+ and P^- be two query patterns, where P^+ is a proper prefix of P^-. We call P^- as the forbidden extension of the included pattern P^+. A forbidden extension query $\langle P^+, P^- \rangle$ asks to report all *occ* documents in $\mathcal{D}$ that contains P^+ as a substring, but does not contain P^- as one. A top-k forbidden extension query $\langle P^+, P^-, k \rangle$ asks to report those k documents among the *occ* documents that are most relevant to P^+. We present a linear index (in words) with an $O(|P^-| + occ)$ query time for the document listing problem. For the top-k version of the problem, we achieve the following results, when the relevance of a document is based on PageRank:

- an $O(n)$ space (in words) index with $O(|P^-| \log \sigma + k)$ query time, where σ is the size of the alphabet from which characters in $\mathcal{D}$ are chosen. For constant alphabets, this yields an optimal query time of $O(|P^-| + k)$.
- for any constant $\epsilon > 0$, a $|\mathsf{CSA}| + |\mathsf{CSA}^*| + D \log \frac{n}{D} + O(n)$ bits index with $O(\mathsf{search}(P) + k \cdot \mathsf{t_{SA}} \cdot \log^{2+\epsilon} n)$ query time, where $\mathsf{search}(P)$ is the time to find the suffix range of a pattern P, $\mathsf{t_{SA}}$ is the time to find suffix (or inverse suffix) array value, and $|\mathsf{CSA}^*|$ denotes the maximum of the space needed to store the *compressed suffix array* CSA of the concatenated text of all documents, or the total space needed to store the individual CSA of each document.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases document retrieval, suffix trees, range queries, succinct data structure

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.320

1 Introduction and Related Work

Retrieving useful information from massive textual data is a core problem in information retrieval. *Document listing*, a natural formulation of this problem, has exciting applications in search engines, bioinformatics, data and Web mining. The task is to index a given collection of strings or documents, such that the relevant documents for an input query can be retrieved efficiently. More formally, let $\mathcal{D}$ be a given collection of D string documents of total size n characters. Given a query pattern P, document listing is to report all the documents that contain P as a substring. The problem was introduced by Matias et al. [19]. Later,

Muthukrishnan [22] proposed a linear space index with optimal query time of $O(|P| + occ)$, where occ is the number of documents reported. Following this, several variations were introduced. Hon et al. [13] proposed the top-k variation i.e., retrieve the k documents that are most relevant to P for some integer k provided at query time They presented a linear index with $O(|P| + k \log k)$ time. Later this was improved to optimal $O(|P| + k)$ time [14, 23]. Compressed indexes have also been proposed for this variation [14, 17, 21, 25].

Most of the earlier document retrieval problems focus on the case where the query consists of a single pattern P. Often the queries are not so simplistic. Muthukrishnan [22] also considered the case of two patterns, say P and Q, and showed that by maintaining an $O(n^{3/2} \log^{O(1)} n)$ space index, documents containing both P and Q can be reported in $O(|P| + |Q| + \sqrt{n} + occ)$ time. Cohen and Porat [4] presented an $O(n \log n)$ space (in words) index with query time $O(|P| + |Q| + \sqrt{n \cdot occ} \log^{5/2} n)$, which was improved by Hon et al. [15] to an $O(n)$ space index with query time $O(|P| + |Q| + \sqrt{n \cdot occ} \log^{3/2} n)$. Also see [14, 13] for a succinct solution and [18] for a recent result on the hardness of this problem.

Fischer et al. [5] introduced the *document listing with forbidden pattern problem* which consists of two patterns P and Q, and all documents containing P but not Q are to be reported. They presented an $O(n^{3/2})$ bit solution with query time $O(|P| + |Q| + \sqrt{n} + occ)$. Hon et al. [16] presented an $O(n)$ word index with query time $O(|P| + |Q| + \sqrt{n \cdot occ} \log^{5/2} n)$. Later, Biswas et al. [1] offered linear space (in words) and $O(|P| + |Q| + \sqrt{nk})$ query time solution for the more general top-k version of the problem, which yields a linear space and $O(|P| + |Q| + \sqrt{n \cdot occ})$ solution to the listing problem. They also showed that this is optimal via a reduction from the *set intersection/difference* problem.

In this paper, we introduce the *document listing with forbidden extension problem*, which is a stricter version of the forbidden pattern problem, and asks to report all documents containing an included pattern P^+, but not its forbidden extension P^-, where P^+ is a proper prefix of P^-. As shown by Biswas et al. [1], the forbidden pattern problem of Fischer et al. [5] suffers from the drawback that linear space (in words) solutions are unlikely to yield a solution better than $O(\sqrt{n/occ})$ per document reporting time. Thus, it is of theoretical interest to see whether this hardness can be alleviated by putting further restrictions on the forbidden pattern. We show that indeed in case when the forbidden pattern is an extension of the included pattern, by maintaining a linear space index, the document listing problem can be answered in optimal $O(|P^-| + occ)$ time. For further theoretical interest, we study the following more general top-k variant.

▶ **Problem 1** (top-k Document Listing with Forbidden Extension). *Let $\mathcal{D} = \mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D$ be D weighted strings (called documents) of n characters in total, where each character is chosen from an alphabet of size σ. Our task is to index $\mathcal{D}$ such that when a pattern P^+, its extension P^-, and an integer k come as a query, among all documents containing P^+, but not P^-, we can report the k most weighted ones.*

Results. Our contributions to Problem 1 are summarized in the following theorems.

▶ **Theorem 1.** *The top-k forbidden extension queries can be answered by maintaining an $O(n)$-words index in $O(|P^-| \log \sigma + k)$ time.*

▶ **Theorem 2.** *Let* CSA *be a* compressed suffix array *on $\mathcal{D}$ of size* $|\mathsf{CSA}|$ *bits using which we can find the suffix range of a pattern P in* search(P) *time, and suffix (or inverse suffix) array value in* t_{SA} *time. Also, denote by* $|\mathsf{CSA}^*|$ *the maximum of the space needed to store the* compressed suffix array CSA *of the concatenated text of all documents, or the total space needed to store the individual* CSA *of each document. By maintaining* CSA *and additional*

$|\mathsf{CSA}^*| + D \log \frac{n}{D} + O(n)$ *bits structure, we can answer* top-k *forbidden extension queries in* $O(\mathsf{search}(P^-) + k \cdot \mathsf{t_{SA}} \cdot \log^{2+\epsilon} n)$ *time*

The rest of the paper is organized as follows. In Section 2, we briefly discuss standard data-structures, and terminologies. In Section 3, we present a linear index and arrive at Theorem 1. In Section 4, we present a succinct space index and arrive at Theorem 2. Finally, we conclude the paper in Section 5.

2 Preliminaries

We refer the reader to [11] for standard definitions and terminologies. We assume the Word-RAM model of computation, where the word size is $\omega = \Theta(\log n)$. Throughout this paper, $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ is a collection of D documents of total size n characters, where each character is chosen from an alphabet of size σ. Each document in $\mathcal{D}$ has a special terminating character that does not appear anywhere in the document. Furthermore, we assume that the PageRank of a document T_d is d, and T_d is more relevant than $\mathsf{T}_{d'}$ iff $d < d'$.

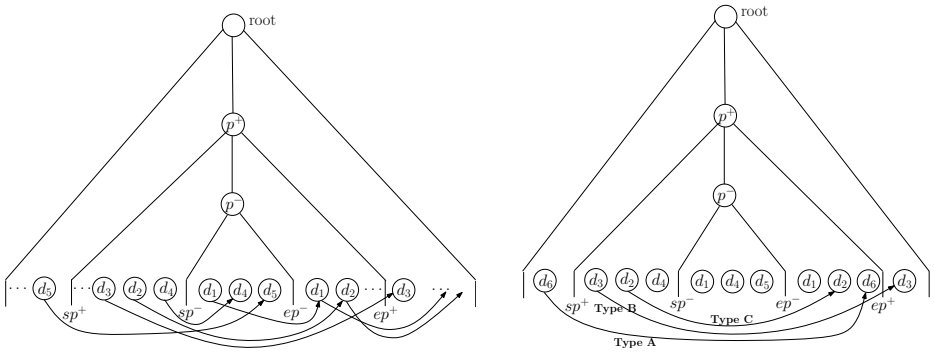
The *generalized suffix tree* GST is a compacted trie that stores all (non-empty) suffixes of every string in $\mathcal{D}$. The GST consists of n leaves and at most $n-1$ internal nodes. We use ℓ_i to denote the ith leftmost leaf of GST i.e., the leaf corresponding to the ith lexicographically smallest suffix of the concatenated text T of every document. Further, $\mathsf{doc}(i)$ denotes the index of the document to which the suffix corresponding to ℓ_i belongs. Let $\mathsf{GST}(u)$ be the sub-tree of GST rooted at u, and $\mathsf{leaf}(u)$ be the set of leaves in $\mathsf{GST}(u)$. We use $\mathsf{leaf}(u, v)$ to denote the leaves in $\mathsf{GST}(u)$ but not in $\mathsf{GST}(v)$. The number of nodes (resp. concatenation of edge labels) on the path from root to a node u is denoted by $\mathsf{depth}(u)$ (resp. $\mathsf{path}(u)$). The *locus* of P, denoted by $\mathsf{locus}(P)$, is the highest node u such that $\mathsf{path}(u)$ is prefixed by P. Then, the *suffix range* of P is $[L_u, R_u]$, where L_u and R_u are the leftmost and the rightmost leaves in $\mathsf{GST}(u)$. By maintaining GST of $\mathcal{D}$ in $O(n)$ words of space, the locus of any pattern P can be computed in $O(|P|)$ time, where $|P|$ is the length of P. In general, suffix trees arrays require $O(n)$ words for storage. *Compressed Suffix Array* reduces this space close to the size of the text with a slowdown in query time.

Let u and v be any two nodes in GST. Then $\mathsf{list}_k(u, v)$ is the set of k most relevant document identifiers in $\mathsf{list}(u, v) = \{\mathsf{doc}(i) \mid \ell_i \in \mathsf{leaf}(u)\} \setminus \{\mathsf{doc}(i) \mid \ell_i \in \mathsf{leaf}(v)\}$. Any superset of $\mathsf{list}_k(u, v)$ is called a k-*candidate set* and is denoted by $\mathsf{cand}_k(u, v)$. Given $\mathsf{cand}_k(u, v)$, we can find $\mathsf{list}_k(u, v)$ in time $O(|\mathsf{cand}_k(u, v)|)$ using order statistics [1].

Moving forward, we use CSA to denote a compressed suffix array for $\mathcal{D}$ that occupies $|\mathsf{CSA}|$ bits. Using CSA, we can find the suffix range of P in $\mathsf{search}(P)$ time, and can compute a suffix array value (i.e., the text position of the suffix corresponding to a leaf) or inverse suffix array value (i.e., the lexicographic rank of a suffix) in $\mathsf{t_{SA}}$ time. Also, p^+ and p^- (resp. $[sp^+, ep^+]$ and $[sp^-, ep^-]$) denotes the loci (resp. suffix ranges) of P^+ and P^- respectively. Since P^- is an extension of P^+, $p^- \in \mathsf{GST}(p^+)$, $\mathsf{leaf}(p^-) \subseteq \mathsf{leaf}(p^+)$, and $sp^+ \leq sp^- \leq ep^- \leq ep^+$.

3 Linear Space Index

In this section, we present our linear space index. We use some well-known range reporting data-structures [2, 24, 26] and the chaining framework of Muthukrishnan [14, 22], which has been extensively used in problems related to document listing.Using these data structures, we first present a solution to the document listing problem. Then, we present a simple linear index for the top-k version of the problem, with a $O(|P^-| \log n + k \log n)$ query time. Using



(a) Chaining structure. **(b)** Types of chains.

Figure 1 Chaining framework. Although $\mathsf{leaf}(p^+)$ has documents d_1, d_2, d_3, d_4, and d_5, only d_2 and d_3 qualify as output, since d_1, d_4, and d_5 are present in $\mathsf{leaf}(p^-)$.

more complicated techniques, based on the heavy path decomposition of a tree, we improve this to arrive at Theorem 1.

Orthogonal Range Reporting Data Structure.

▶ **Fact 1** ([24]). *A set of n weighted points on an $n \times n$ grid can be indexed in $O(n)$ words of space, such that for any $k \geq 1$, $h \leq n$ and $1 \leq a \leq b \leq n$, we can report k most weighted points in the range $[a, b] \times [0, h]$ in decreasing order of their weights in $O(h + k)$ time.*

▶ **Fact 2** ([26]). *A set of n 3-dimensional points (x, y, z) can be stored in an $O(n)$-word data structure, such that we can answer a three-dimensional dominance query in $O(\log n + output)$ time, with outputs reported in the sorted order of z-coordinate.*

▶ **Fact 3** ([2]). *Let A be an array of length n. By maintaining an $O(n)$-words index, given two integers i, j, where $j \geq i$, and a positive integer k, in $O(k)$ time, we can find the k largest (or, smallest) elements in the subarray $A[i..j]$ in sorted order.*

Chaining Framework. For every leaf ℓ_i in GST, we define $\mathsf{next}(i)$ as the minimum index $j > i$, such that $\mathsf{doc}(j) = \mathsf{doc}(i)$. We denote i as the source of the chain and $\mathsf{next}(i)$ as the destination of the chain. We denote by $(-\infty, i)$ (resp. (i, ∞)) the chain that ends (resp. starts) at the first (resp. last) occurrence ℓ_i of a document. Figure 1(a) illustrates chaining.

The integral part of our solution involves categorizing the chains into the following 3 types, and then build separate data structure for each type.

Type A: $i < sp^+$ and $ep^- < \mathsf{next}(i) \leq ep^+$

Type B: $sp^+ \leq i < sp^-$ and $\mathsf{next}(i) > ep^+$

Type C: $sp^+ \leq i < sp^-$ and $ep^- < \mathsf{next}(i) \leq ep^+$

Figure 1(b) illustrates different types of chains. It is easy to see that any output of forbidden extension query falls in one of these 3 types. Also observe that the number of chains is n. For a type A chain $(i, \mathsf{next}(i))$, we refer to the leaves ℓ_i and $\ell_{\mathsf{next}(i)}$ as type A leaves; similar remarks hold for type B and type C chains. Also, LCA of a chain (i, j) refers to the LCA of the leaves ℓ_i and ℓ_j. Furthermore, with slight abuse of notation, for any two nodes $u, v \in \mathsf{GST}$, we denote by $\mathsf{depth}(u, v)$, the depth of the LCA of the nodes u and v.

Document Listing Index. Linear space index for the forbidden extension document listing problem is achieved by using Fact 3. We store two arrays as defined below.
A_{src}: $A_{src}[i]$=$\mathsf{next}(i)$, for each chain $(i, \mathsf{next}(i))$
A_{dest}: $A_{dest}[\mathsf{next}(i)]$=$i$, for each chain $(i, \mathsf{next}(i))$
Querying in A_{src} within the range $[sp^+, sp^- - 1]$ will give us the chains in descending order of their destination, we stop at ep^- to obtain all the Type B and Type C chains. We query in A_{dest} within the range $[ep^- + 1, ep^+]$ to obtain the chains in ascending order of their source and stop at sp^+ to obtain all the type A chains. Time, in addition to that required for finding the suffix ranges, can be bounded by $O(|P^-| + occ)$.

3.1 A Simple $O(|P^-| \log n + k \log n)$ time Index

We start with a simple indexing scheme for answering top-k forbidden extension query. In this section, we design data structures by processing different types of chains separately and mapping them into range reporting problem.

Processing Type A and Type B Chains. For type A chains, we construct range reporting data structure, as described in Fact 1, with each chain (i, j), $j = \mathsf{next}(i)$, mapped to a weighted two dimensional point $(j, \mathsf{depth}(i, j))$ with weight $\mathsf{doc}(i)$. Likewise, for type B chains, we map chain (i, j) to the point $(i, \mathsf{depth}(i, j))$ with weight $\mathsf{doc}(i)$. Recall that d is the PageRank of the document T_d. For Type A chains, we issue a range reporting query for $[ep^- + 1, ep^+] \times [0, \mathsf{depth}(p^+)]$. For Type B chains, we issue a range reporting query for $[sp^+, sp^- - 1] \times [0, \mathsf{depth}(p^+)]$. In either case, we can obtain the top-k leaves in sorted order of their weights in $O(|P^-| + k)$ time, which gives us the following lemma.

▶ **Lemma 3.** *There exists an $O(n)$ words data structure, such that for a top-k forbidden extension query, we can report the top-k Type A and Type B leaves in time $O(|P^-| + k)$.*

Processing Type C Chains. We maintain the 3-dimensional dominance structure of Fact 2 at each node of GST. For a chain (i, j), $j = \mathsf{next}(i)$, we store the point $(i, j, \mathsf{doc}(i))$ in the dominance structure maintained in the node $\mathsf{lca}(i, j)$. For query answering, we traverse the path from p^+ to p^-, and query the dominance structure of each node on this path with x-range $[-\infty, sp^- - 1]$ and y-range $[ep^- + 1, \infty]$. Any chain falling completely outside of $\mathsf{GST}(p^+)$ will not be captured by the query, since their LCA lies above p^+. There can be at most $\mathsf{depth}(p^-) - \mathsf{depth}(p^+) + 1 \leq |P^-| = \Theta(n)$ sorted lists containing k elements each. The $\log n$ factor in the query of Fact 2 is due to locating the first element to be extracted; each of the remaining $(k - 1)$ elements can be extracted in constant time per element. Therefore, time required for dominance queries (without extracting the elements) is bounded by $O(|P^-| \log n)$. Using a max-heap of size $O(n)$, we obtain the top-k points from all the lists as follows: insert the top element from each list into the heap, and extract the maximum element from the heap. Then, the next element from the list corresponding to the extracted element is inserted into the heap. Clearly, after extracting k elements, the desired top-k identifiers are obtained. Time required is $O(k \log n)$, which gives the following lemma.

▶ **Lemma 4.** *There exists a $O(n)$ words space data-structure for answering top-k documents with forbidden extension queries in $O(|P^-| \log n + k \log n)$ time.*

3.2 $O(|P^-| \log \sigma + k)$ Index

In this section, we prove Theorem 1. Note that type A and type B chains can be processed in $O(|P^-| + k)$ time by maintaining separate range reporting data structures (refer to

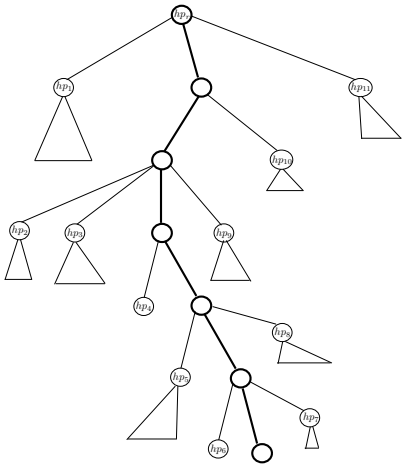
Section 3.1). Therefore, in what follows, the emphasis is to obtain type C outputs. Recall that for processing type C chains in Section 3.1, we traversed the path from p^+ to p^-, and query the individual data structure at each node. Our idea for more efficient solution is to group together the data structures of the nodes falling on the same heavy path.

Heavy Path Decomposition. We revisit the heavy path decomposition of a tree T, proposed by Harel et al. [12]. For any internal node u, the heaviest child of u is the one having the maximum number of leaves in its subtree (ties broken arbitrarily). The first heavy path of T is the path starting at T's root, and traversing through every heavy node to a leaf. Each off-path subtree of the first heavy path is further decomposed recursively. Thus, a tree with m leaves has m heavy paths. With slight abuse of notation, let $\mathsf{leaf}(hp_i)$ be the leaf where heavy path hp_i ends. Let v be a node on a heavy path and u be a child of v not on that heavy path. We say that the subtree rooted at u *hangs from node* v.

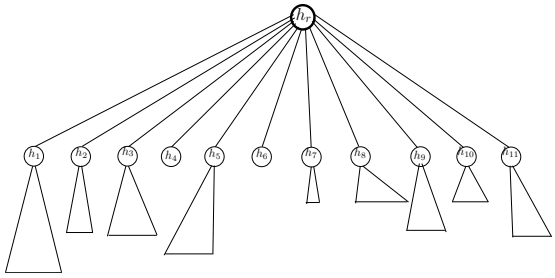
▶ **Property 1.** *For a tree having m nodes, the path from the root to any node v traverses at most $\log m$ heavy paths.*

Heavy Path Tree. We construct the heavy path tree T_H, in which each node corresponds to a distinct heavy path in GST. The tree T_H has n nodes as there are so many heavy paths in GST. For a heavy path hp_i of GST, the corresponding node in T_H is denoted by h_i. All the heavy paths hanging from hp_i in GST are the children of h_i in T_H. Let the first heavy path in the heavy path decomposition of GST be hp_r, and $T_1, T_2, \ldots,$ be the subtrees hanging from hp_r. The heavy path tree T_H is recursively defined as the tree whose root is h_r, representing hp_r, having children $h_1, h_2, \ldots$ with subtrees in T_H resulting from the heavy path decomposition of $T_1, T_2, \ldots$ respectively. Figure 2 illustrates heavy path decomposition of GST and the heavy path tree T_H. Based on the position of a hanging heavy path w.r.t. hp_i in GST, we divide the children of h_i into two groups: left children h_i^l and right children h_i^r. A child heavy path h_j of h_i belongs to h_i^l (resp. h_i^r) if $\mathsf{leaf}(hp_j)$ falls on the left (resp. right) of $\mathsf{leaf}(hp_i)$ in GST. The nodes in h_i^l and h_i^r are stored contiguously in T_H. We traverse the left attached heavy paths of hp_i in GST in top-to-bottom order, include them as the nodes of h_i^l, and place them in left-to-right order as children of h_i in T_H. The h_i^r nodes are obtained by traversing the right attached heavy paths of hp_i in GST in bottom-to-top order, and place them after the h_i^l nodes in T_H in left-to-right order.

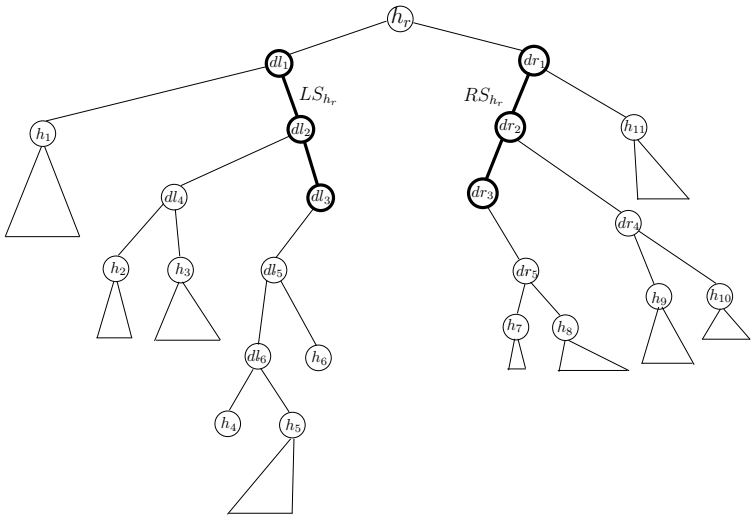
Transformed Heavy Path Tree. We transform the heavy path tree T_H into a binary search tree T_H^t. For each node h_i in T_H, we construct a left (resp. right) binary tree $BT_{h_i^l}$ (resp. $BT_{h_i^r}$) for the left children h_i^l (resp. right children h_i^r). Leaves of $BT_{h_i^l}$ (resp. $BT_{h_i^r}$) are the nodes of h_i^l (resp. h_i^r) preserving the ordering in T_H. The binary tree $BT_{h_i^l}$ (resp. $BT_{h_i^r}$) has a path, named left spine (resp. right spine), denoted by LS_{h_i} (resp. RS_{h_i}) containing $\lfloor \log |h_i^l| \rfloor$ (resp. $\lfloor \log |h_i^r| \rfloor$) nodes, denoted by $dl_1, dl_2, \ldots$ (resp. $dr_1, dr_2, \ldots$) in the top-to-bottom order. The right child of dl_i is dl_{i+1}. Left subtree of dl_i is a height balanced binary search tree containing $h_{2^{i-1}}, \ldots, h_{\lfloor 2^i - 1 \rfloor}$ as the leaves and dummy nodes for binarization. Right spine is constructed in similar way, however left child of dr_i is dr_{i+1} and left subtree contains the leaves of h_i^r in a height balanced binary tree. Clearly, the length of LS_{h_i} (resp. RS_{h_i}) is bounded by $\lfloor \log |h_i^l| \rfloor$ (resp. $\lfloor \log |h_i^r| \rfloor$). Subtrees hanging from the nodes of h_i^l and h_i^r are decomposed recursively. See Figure 2(c) for illustration. We have the following important property of T_H^t.



(a) Heavy path decomposition of GST.



(b) Heavy path tree T_H.



(c) Transformed heavy path tree T_H^t.

Figure 2 Heavy path decomposition, heavy path tree, and transformed heavy path tree.

▶ **Lemma 5.** *Let u be an ancestor node of v in* GST*. The path length from u to v is d_{uv}. The node u (resp. v) falls on the heavy path hp_1 (resp. hp_t) and let h_1 (resp. h_t) be the corresponding node in T_H^t. Then, the h_1 to h_t path in T_H^t has $O(\min(d_{uv} \log \sigma, \log^2 n))$ nodes, where σ is the size of the alphabet from which characters in the documents are chosen.*

Proof. We first recall from Property 1 that the height of T_H is $O(\log n)$. Since each node in T_H can have at most n children, each level of T_H can contribute to $O(\log n)$ height in T_H^t. Thus, the height of T_H^t is bounded by $O(\log^2 n)$. Hence, the $\log^2 n$ bound in the lemma is immediate. Let $p_1, p_2, \ldots, p_t$ be the segments of the path from u to v traversing heavy paths $hp_1, hp_2, \ldots, hp_t$, where $p_i \in hp_i, 1 \leq i \leq t$. Let $h_1, h_2, \ldots, h_t$ be the corresponding nodes in T_H^t. We show that the number of nodes traversed to reach from h_i to h_{i+1} in T_H^t is $O(|p_i| \log \sigma)$. Without loss of generality, assume h_{i+1} is attached on the left of of h_i and falls in the subtree attached with dl_x on spine LS_{h_i}. We can skip all the subtrees attached to the nodes above dl_x on LS_{h_i}. One node on a heavy path can have at most σ heavy paths

as children. Thus, the number of nodes traversed on the spine is $O(|p_i| \log \sigma)$. Within the subtree of the dl_x, we can search the tree to find the desired heavy path node. Since, each node in the GST can have at most σ heavy paths as children, the height of this subtree is bounded by $O(\log \sigma)$. For each p_i, we may need to traverse the entire tree height to locate the desired heavy path, and hence the lemma follows. ◀

Associating the chains. Let hp_i (resp. hp_j) be the heavy path having i (resp. j) as the leaf node in GST and h_i (resp. h_j) as the corresponding heavy path node in T_H^t. Then, we *associate* chain (i, j) with $\mathsf{lca}(h_i, h_j)$ in T_H^t.

Constructing the Index. Our index consists of two components, maximum chain depth structure (MDS) and transformed heavy path structure (THS) defined as follows.

MDS component: Let hp_t be the heavy path in the original heavy path decomposition (i.e., not a dummy heavy path), associated with chain (i, j), $j = \mathsf{next}(i)$. Let, $d_i = \mathsf{depth}(i, \mathsf{leaf}(hp_t))$ and $d_j = \mathsf{depth}(j, \mathsf{leaf}(hp_t))$. Define $\mathsf{maxDepth}(i, j) = \max(d_i, d_j)$. Let m_t be the number of chains associated with hp_t. Create two arrays A_t and A'_t, each of length m_t. For each chain (i, j) associated with hp_t, store $\mathsf{doc}(i)$ in the first empty cell of the array A_t, and $\mathsf{maxDepth}(i, j)$ in the corresponding cell of the array A'_t. Sort both the arrays w.r.t the values in A'_t. For each node u lying on hp_t, maintain a pointer to the minimum index x of A such that $A'_t[x] = \mathsf{depth}(u)$. Discard the array A'_t. Finally, build the 1-dimensional sorted range-reporting structure (Fact 3) over A_t. Total space for all t is bounded by $O(n)$ words.

THS component: We construct the transformed heavy path tree T_H^t from GST. Recall that every chain in GST is associated with a node in T_H^t. For each node h_i in T_H^t, we store two arrays, chain source array CS_i and chain destination array CD_i. The arrays CS_i (resp. CD_i) contains the weights (i.e., the document identifier) of all the chains associated with h_i sorted by the start (resp. end) position of the chain in GST. Finally we build the RMQ data structure (Fact 4) RMQ_{CS_i} and RMQ_{CD_i} over CS_i and CD_i respectively. Total space can be bounded by $O(n)$ words.

▶ **Fact 4** ([6, 7]). *By maintaining a $2n+o(n)$ bits structure, range maximum query(RMQ) can be answered in $O(1)$ time (without accessing the array).*

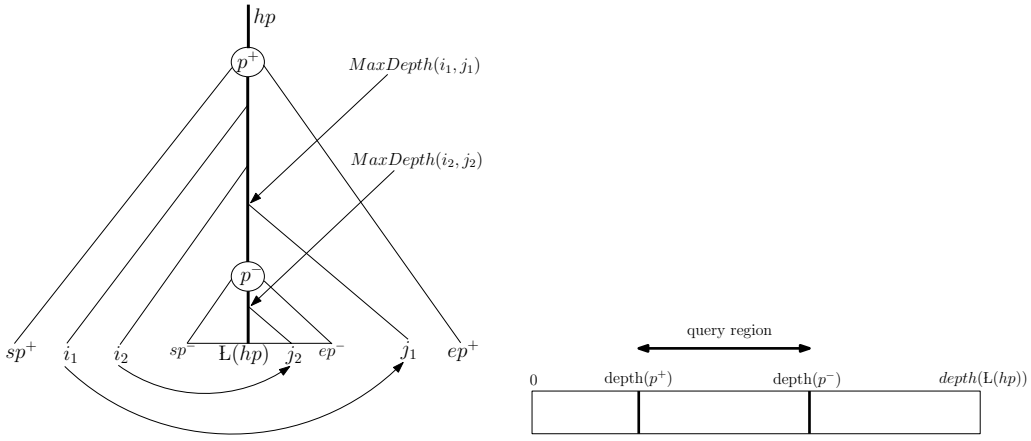
Query Answering. Query answering is done by traversing from p^+ to p^- in GST. We start with the following observation.

▶ **Observation 1.** *For every type C chain (i, j), $\mathsf{lca}(i, j)$ falls on the p^+ to p^- path in* GST.

This observation is crucial to ensure that we do not miss any type C chain in query answering. We consider the following two cases for query answering.

3.2.1 p^+ and p^- falls on the same heavy path

In this case, we resort to component MDS for query answering. Assume that p^+ and p^- fall on heavy path hp_t. Note that a chain (i, j) qualifies as an output, iff $\mathsf{maxDepth}(i, j)$ falls within the range $[\mathsf{depth}(p^+), \mathsf{depth}(p^-) - 1]$. See Figure 3(a) for illustration. For query answering, follow the pointers from p^+ and p^- to the indexes x and y in the array A_t, and issue the query $\langle x, y - 1, k\rangle$ in the corresponding Fact 3 data structure. Note that Type A and Type B outputs can arise. We obtain the following lemma.



(a) Chain (i_1, j_1) qualifies since $\mathsf{maxDepth}(i, j) \in [\mathsf{depth}(p^+), \mathsf{depth}(p^-))$. (i_2, j_2) does not qualify.

(b) Query range in the 1-dimensional sorted range reporting structure of hp.

Figure 3 p^+ and p^- falling on the same heavy path.

▶ **Lemma 6.** *There exists an $O(n)$ words data structure, such that for a* top-k *forbidden extension query, we can report the* top-k *Type C leaves in $O(|P^-| + k)$ time when p^+ and p^- falls on the same heavy path.*

3.2.2 p^+ and p^- falls on different heavy paths

Let $p_1, p_2, \ldots, p_t$ be the path segments of the path from p^+ to p^- traversing heavy paths $hp_1, hp_2, \ldots, hp_t$, where $p_i \in hp_i$, $1 \leq i \leq t$. Let $h_1, h_2, \ldots, h_t$ be the corresponding nodes in T_H^t. In the following subsection, we show how to obtain answers for h_1 through h_{t-1}; we resolve h_t separately. We use the THS component for processing the chains with LCA on $h_1, h_2, \ldots, h_{t-1}$. We start with the following lemma.

▶ **Lemma 7.** *Let (i, j) be a chain associated with a node h_k in T_H^t. If p^- falls on the left (resp. right) subtree of h_k, and $sp^+ \leq i < sp^-$ (resp. $ep^- < j \leq ep^+$), then (i, j) is qualified as an output of the forbidden extension query.*

Proof. Recall that chain (i, j) is associated with $h_k = \mathsf{lca}(h_i, h_j)$ in T_H^t, where h_i and h_j are the heavy path nodes corresponding to i and j respectively. This implies h_i (resp. h_j) falls on the left (resp. right) subtree of h_k. If p^- falls on the left of hp_k then $j > ep^-$. The added constraint $sp^+ \leq i < sp^-$ ensures that chain (i, j) is either a Type B or a Type C chain, both of which are qualified as an output of the forbidden extension query. The case when p^- falls on the right of h_k is symmetric. ◀

Lemma 7 allows us to check only the source or destination of a chain based on the position of p^-, and collect the top weighted chains; this is facilitated using the RMQ data structure. We traverse the nodes in T_H^t from p^+ to p^-. At each node h_k, if p^- falls on the left of h_k, we issue a range maximum query within the range $[sp^+, sp^- - 1]$ on RMQ_{CS_k} which gives us the top answer from each node in $O(1)$ time. Note that, $[sp^+, sp^- - 1]$ range needs to be transformed for different RMQ_{CS} structures. We use fractional cascading for the range transformation to save predecessor searching time (refer to Appendix A for detailed discussion). Since the height of the tree is $O(\log^2 n)$ (refer to Lemma 5) at any instance, there are at most $O(\log^2 n)$ candidate points. We use the *atomic heap* of Fredman and Willard [9] which allows constant

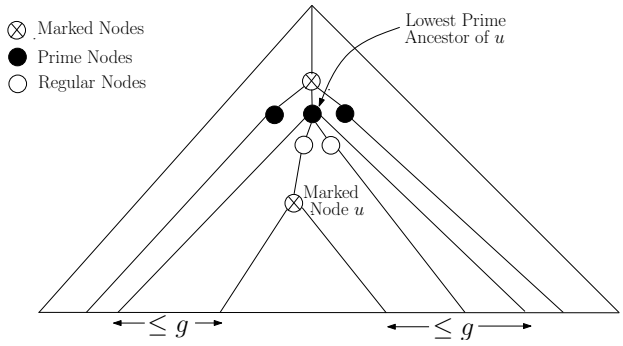


Figure 4 Marked nodes and Prime nodes with respect to grouping factor g.

time insertion and delete-max operation when the heap size is $O(\log^2 m)$, where m is the size of the universe. By maintaining each candidate point in the atomic heap, the highest weighted point (among all candidate points) can be obtained in constant time. Also, once the highest weighted point from a heavy path node is obtained, each subsequent candidate point can be obtained and inserted into the the atomic heap in $O(1)$ time. Hence the query time is bounded by the number of nodes traversed in T_H^t. From lemma 5, we obtain that the number of nodes traversed is bounded by $O(\min(|P^-| \log \sigma, \log^2 n))$.

For hp_t, we utilize component MDS. Let r_t be the root of heavy path hp_t. A chain (i, j) qualifies as an output, iff $\mathsf{maxDepth}(i, j)$ falls within the range $[\mathsf{depth}(r_t), \mathsf{depth}(p^-) - 1]$. For query answering, follow the pointers from r_t and p^- to the indexes x and y in the array A_t, and issue the query $\langle x, y - 1, k\rangle$ in the corresponding Fact 3 data structure. Note that Type A and Type B outputs can arise.

From the above discussion, we obtain the following lemma.

▶ **Lemma 8.** *There exists an $O(n)$ words data structure, such that for a* top-k *forbidden extension query, we can report the* top-k *Type C leaves in $O(|P^-| \log \sigma + k)$ time when p^+ and p^- falls on different heavy paths.*

Combining Lemmas 3, 6, and 8, we obtain the result stated in Theorem 1.

4 Succinct Index

In this section, we prove Theorem 2. The key idea is to identify some special nodes in the GST, pre-compute the answers for a special node and its descendant special node, and maintain these answers in a data structure. By appropriately choosing the special nodes, the space can be bounded by $O(n)$ bits. Using other additional compressed data structures for document listing [14], we arrive at our claimed result.

We begin by identifying certain nodes in GST as *marked nodes* and *prime nodes* based on a parameter g called *grouping factor* [13]. First, starting from the leftmost leaf in GST, we combine every g leaves together to form a group. In particular, the leaves ℓ_1 through ℓ_g forms the first group, ℓ_{g+1} through ℓ_{2g} forms the second, and so on. We mark the LCA of the first and last leaves of every group. Moreover, for any two marked nodes, we mark their LCA (and continue this recursively). Note that the root node is marked, and the number of marked nodes is at most $2\lceil n/g \rceil$. See Figure 4 for an illustration.

Corresponding to each marked node (except the root), we identify a unique node called the prime node. Specifically, the prime node u' corresponding to a marked node u^* is the

node on the path from root to u^*, which is a child of the lowest marked ancestor of u^*; we refer to u' as the lowest prime ancestor of u^*. Since the root node is marked, there is always such a node. If the parent of u^* is marked, then u^* is same as u'. Also, for every prime node, the corresponding closest marked descendant (and ancestor) is unique. Therefore number of prime nodes is one less than the number of marked nodes. The following lemma highlights some important properties of marked and prime nodes.

▶ **Fact 5** ([1, 14]). *(i) In constant time we can verify whether any node has a marked descendant or not. (ii) If a node u has no marked descendant, then $|\mathsf{leaf}(u)| < 2g$. (iii) If u^* is the highest marked descendant of u, and u is not marked, then $|\mathsf{leaf}(u, u^*)| \leq 2g$. (iv) If u' is the lowest prime ancestor of u^*. Then $|\mathsf{leaf}(u', u^*)| \leq 2g$.*

We now present a framework for proving the following lemma.

▶ **Lemma 9.** *Assume the following.*

(a) *The highest marked node u^* and the sequence of prime nodes (if any) on the path from p^+ to p^- can be found in $\mathsf{t}_{\mathsf{prime}}$ time.*

(b) *For any leaf ℓ_i, we can find the corresponding document in t_{DA} time.*

(c) *For any document identifier d and a range of leaves $[sp, ep]$, we can check in $\mathsf{t}_{\in}$ time, whether d belongs in $\{\mathsf{doc}(i) \mid sp \leq i \leq ep\}$, or not.*

For any function $f(n)$, such that $f(n) = \Omega(1)$ and $f(n) = o(n)$, by maintaining CSA *and additional $O((n/f(n)) \log^2 n)$ bits structures, we can answer* top-k *forbidden extension queries in $O(\mathsf{search}(P^-) + \mathsf{t}_{\mathsf{prime}} + k \cdot f(n) \cdot (\mathsf{t}_{\mathsf{DA}} + \mathsf{t}_{\in}))$ time.*

Creating the Index. First we maintain a full-text index CSA on the document collection $\mathcal{D}$. Let $g_\kappa = \lceil \kappa \cdot f(n) \rceil$, where κ is a parameter to be defined later. We begin by marking nodes in the GST as marked and prime nodes, as defined previously, based on g_κ. Consider any prime node u, and let $u^\uparrow$ and $u^\downarrow$ be its nearest marked ancestor and descendant (both of which are unique) respectively. We compute the arrays $\mathsf{list}_\kappa(u^\uparrow, u)$ and $\mathsf{list}_\kappa(u, u^\downarrow)$, each sorted by increasing importance (i.e., document identifier). The arrays are maintained in the node u w.r.t grouping factor g_κ. Note that explicitly maintaining each array requires $O(\kappa \log n)$ bits. Space required in bits for all prime nodes w.r.t g_κ can be bounded by $O((n/g_\kappa)\kappa \log n)$ i.e., by $O((n/f(n)) \log n)$ bits. We maintain this data-structure for $\kappa = 1, 2, 4 \ldots, D$. Total space is bounded by $O((n/f(n)) \log^2 n)$ bits.

Querying Answering. For a top-k forbidden extension query $\langle P^+, P^-, k \rangle$, we begin by locating the suffix ranges $[sp^+, ep^+]$ and $[sp^-, ep^-]$ of the patterns P^+ and P^- respectively; this can be achieved in time bounded by $\mathsf{search}(P^-)$ using the CSA. If the suffix ranges are the same, then clearly every document containing P^+ also contains P^-, and the top-k list is empty. So, moving forward, we assume otherwise. Note that it suffices to obtain a k-candidate set of size $O(k \cdot f(n))$ in the time of Lemma 9.

Let $k' = \min\{D, 2^{\lceil \log k \rceil}\}$. Note that $k \leq k' < 2k$. Moving forwards, we talk of prime and marked nodes w.r.t grouping factor $g' = \lceil k' f(n) \rceil$. We can detect the presence of marked nodes below p^+ and p^- in constant time using Fact 5. Let the prime nodes on the path from p^+ to p^- be $u_1, u_2, \ldots, u_t$ in order of depth. Possibly, $t = 0$. For each prime node $u_{t'}$, $1 \leq t' \leq t$, we denote by $u_{t'}^\uparrow$ and $u_{t'}^\downarrow$, the lowest marked ancestor (resp. highest marked descendant) of the $u_{t'}$. We have the following cases.

▶ **Case 1.** *We consider the following two scenarios: (i) $\mathsf{GST}(p^+)$ does not contain any marked node, and (ii) $\mathsf{GST}(p^+)$ contains a marked node, but the path from p^+ to p^- does*

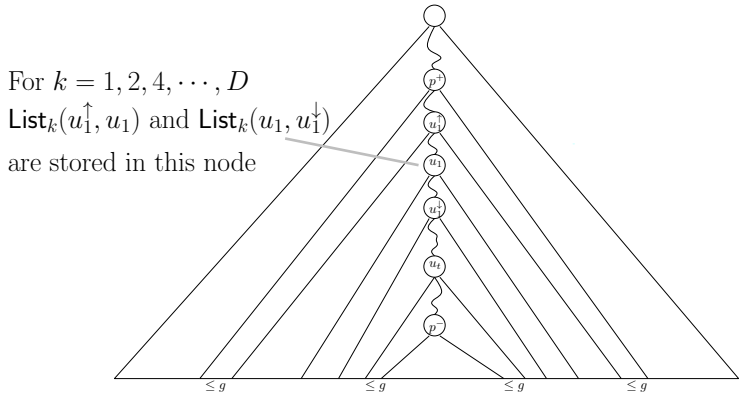


Figure 5 Illustration of storage scheme and retrieval at every prime node w.r.t grouping factor g. Left and right fringes in $\mathsf{leaf}(p^+ \setminus u_1^{\uparrow})$ and $\mathsf{leaf}(u_t^{\downarrow} \setminus p^-)$ are bounded above by g'.

not contain any prime node. In either case, $|\mathsf{leaf}(p^+, p^-)| \leq 2g'$ *(refer to Fact 5). The documents corresponding to these leaves constitute a* k*-candidate set, and can be found in* $O(g' \cdot \mathsf{t_{DA}})$ *time i.e., in* $O(k \cdot f(n) \cdot \mathsf{t_{DA}})$ *time. Now, for each document* d*, we check whether* $d \in \{\mathsf{doc}(i) \mid i \in [sp^-, ep^-]\}$*, which requires additional* $O(g' \cdot \mathsf{t}_{\in})$ *time. Total time can be bounded by* $O(g' \cdot (\mathsf{t_{DA}} + \mathsf{t}_{\in}))$ *i.e., by* $O(k \cdot f(n) \cdot (\mathsf{t_{DA}} + \mathsf{t}_{\in}))$*.*

▶ **Case 2.** *If the path from* p^+ *to* p^- *contains a prime node, then let* u^* *be the highest marked node. Possibly,* $u^* = p^+$*. Note that* $u_1^{\uparrow}$ *is same as* u^**, and that* $u_t^{\downarrow}$ *is either* p^- *or a node below it. For any* t'*, clearly* $\mathsf{list}_{k'}(u_{t'}, u_{t'}^{\downarrow})$ *and* $\mathsf{list}_{k'}(u_{t'}^{\uparrow}, u_{t'})$ *are mutually disjoint. Similar remarks hold for the lists stored at two different prime nodes* t' *and* t''*,* $1 \leq t', t'' \leq t$*. Furthermore, let* d *be an identifier in one of the lists corresponding to* $u_{t'}$*. Clearly there is no leaf* $\ell_j \in \mathsf{GST}(p^-)$*, such that* $\mathsf{doc}(j) = d$*. We select the* top-k' *document identifiers from the stored lists (arrays) in the prime nodes* u_1 *through* u_t*. Time, according to the following fact, can be bounded by* $O(t + k)$*.*

▶ **Fact 6** ([2, 8])**.** *Given* m *sorted integer arrays, we can find the* k *largest values from all these arrays in* $O(m + k)$ *time.*

Now, we consider the fringe leaves $\mathsf{leaf}(p^+, u^*)$ *and* $\mathsf{leaf}(u_t, p^-)$*, both of which are bounded above by* $2g'$ *(refer to Fact 5). The ranges of the these leaves are found in constant time using the following result of Sadakane and Navarro [27].*

▶ **Lemma 10** ([27])**.** *An* m *node tree can be maintained in* $O(m)$ *bits such that given a node* u*, we can find* $[sp(u), ep(u)]$ *in constant time.*

The relevant documents corresponding to these fringe leaves can be retrieved as in Case 1. Clearly, these fringe documents along with the k *documents obtained from the stored lists constitute our* k*-candidate set. Time required can be bounded by* $O(t + k + g' \cdot (\mathsf{t_{DA}} + \mathsf{t}_{\in}))$ *i.e, by* $O(t + k \cdot f(n) \cdot (\mathsf{t_{DA}} + \mathsf{t}_{\in}))$*.*

Note that $t \leq \mathsf{depth}(p^-) \leq |P^-| = O(\mathsf{search}(P^-))$, and Lemma 9 follows. ◀

We are now equipped to prove Theorem 2. First, the highest marked node and the t prime nodes from p^+ to p^- are obtained using Lemma 11 in $O(\log n + t)$ time. Maintain the data-structure of this lemma for with $\kappa = 1, 2, 4, \ldots, D$. Space can be bounded by $O(\frac{n}{f(n)} \log n)$

bits. Computing $\mathsf{doc}(i)$ is achieved in $\mathsf{t_{SA}}$ time, according to Lemma 12. Checking whether a document d belongs in a contiguous range of leaves is achieved in $O(\mathsf{t_{SA}} \cdot \log\log n)$ using Lemma 13. Theorem 2 is now immediate by choosing $f(n) = \log^2 n$.

▶ **Lemma 11.** *By maintaining $O((n/g_\kappa)\log n)$ bits in total, we can retrieve the highest marked node, and all t prime nodes, both w.r.t grouping factor $g_\kappa = \lceil \kappa \cdot f(n) \rceil$, that lie on the path from p^+ to p^- in time bounded by $O(\log n + t)$.*

Proof. We use the following result of Patil et al. [26]: a set of n three-dimensional points (x, y, z) can be stored in an $O(n \log n)$ bits data structure, such that for a three-dimensional dominance query $\langle a, b, c\rangle$, in $O(\log n + t)$ time, we can report all t points (x, y, z) that satisfies $x \le a$, $y \ge b$, and $z \ge c$ with outputs reported in the sorted order of z coordinate.

For each prime node w, we maintain the point $(L_w, R_w, |\mathsf{path}(w)|)$ in the data structure above, where L_w and R_w are the leftmost and the rightmost leaves in $\mathsf{GST}(w)$. Total space in bits can be bounded by $O((n/g_\kappa)\log n)$ bits. The t prime nodes that lie on the path from p^+ to p^- are retrieved by querying with $\langle sp^- - 1, ep^- + 1, |P^+|\rangle$. Time can be bounded by $O(\log n + t)$. Likewise, we maintain a structure for marked nodes. Using this, we can obtain the highest marked node in $O(\log n)$ time. ◀

▶ **Lemma 12.** *Given a* CSA*, the document array can be maintained in additional $n + o(n)$ bits such that for any leaf ℓ_i, we can find $\mathsf{doc}(i)$ in $\mathsf{t_{SA}}$ time i.e., $\mathsf{t_{DA}} = \mathsf{t_{SA}}$.*

Proof. We use the following data-structure [10, 20]: a bit-array $\mathsf{B}[1 \ldots m]$ can be encoded in $m + o(m)$ bits, such that $\mathsf{rank_B}(q, i) = |\{j \in [1..i] \mid \mathsf{B}[j] = q\}|$ can be found in $O(1)$ time.

Consider the concatenated text T of all the documents which has length n. Let B be a bit array of length n such that $\mathsf{B}[i] = 1$ if a document starts at the position i in the text T. We maintain a rank structure on this bit-array. Space required is $n + o(n)$ bits. We find the text position j of ℓ_i in $\mathsf{t_{SA}}$ time. Then $\mathsf{doc}(i) = \mathsf{rank_B}(1, j)$, and is retrieved in constant time. Time required can be bounded by $\mathsf{t_{SA}}$. ◀

▶ **Lemma 13.** *Given the suffix range $[sp, ep]$ of a pattern P and a document identifier d, by maintaining* CSA *and additional $|\mathsf{CSA}^*| + D \log \frac{n}{D} + O(D) + o(n)$ bits structures, in $O(\mathsf{t_{SA}} \log\log n)$ time we can verify whether $d \in \{\mathsf{doc}(i) \mid i \in [sp, ep]\}$, or not.*

Proof. Number of occurrences of d in a suffix range $[sp, ep]$ is given by $\mathsf{rank_{DA}}(d, ep) - \mathsf{rank_{DA}}(d, sp - 1)$. Space and time complexity is due to the following result of Hon et al. [14]: the document array DA can be simulated using CSA and additional $|\mathsf{CSA}^*| + D \log \frac{n}{D} + O(D) + o(n)$ bits structures to support $\mathsf{rank_{DA}}$ operation in $O(\mathsf{t_{SA}} \log\log n)$ time. ◀

5 Concluding Remarks

In this paper, we introduce the problem of top-k forbidden extension query, and propose a linear space index for answering such queries. By maintaining a linear space index, the general forbidden pattern query for an included pattern P, and a forbidden pattern Q, can be answered in $O(|P| + |Q| + \sqrt{n \cdot occ})$ time, where occ is the number of documents reported. We show that by maintaining a linear space index, we can answer forbidden extension queries in optimal $O(|P^-| + occ)$ time. We also address the more general top-k version of the problem, where the relevance measure is based on PageRank. We show that by maintaining linear space index, we obtain a query time of $O(|P^-| \log \sigma + k)$, which is optimal for constant alphabets. Furthermore, we obtain a succinct solution to this problem.

References

1 Sudip Biswas, Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Ranked document retrieval with forbidden pattern. In Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, editors, *Combinatorial Pattern Matching – 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 – July 1, 2015, Proceedings*, volume 9133 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2015.

2 Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro López-Ortiz. Online sorted range reporting. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2009.

3 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

4 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theoretical Computer Science*, 411(40–42):3795–3800, 2010.

5 Johannes Fischer, Travis Gagie, Tsvi Kopelowitz, Moshe Lewenstein, Veli Mäkinen, Leena Salmela, and Niko Välimäki. Forbidden patterns. In *Proceedings of the 10th Latin American International Conference on Theoretical Informatics*, LATIN'12, pages 327–337, Berlin, Heidelberg, 2012. Springer-Verlag.

6 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006.

7 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007.

8 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in X+Y and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982.

9 Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

10 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 368–373. ACM Press, 2006.

11 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997.

12 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

13 Wing-Kai Hon, R. Shah, and J.S. Vitter. Space-efficient framework for top-k string retrieval problems. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 713–722, Oct 2009.

14 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *J. ACM*, 61(2):9, 2014.

15 Wing-Kai Hon, Rahul Shah, SharmaV. Thankachan, and JeffreyScott Vitter. String retrieval for multi-pattern queries. In Edgar Chavez and Stefano Lonardi, editors, *String*

Processing and Information Retrieval, volume 6393 of *Lecture Notes in Computer Science*, pages 55–66. Springer Berlin Heidelberg, 2010.

16 Wing-Kai Hon, Rahul Shah, SharmaV. Thankachan, and JeffreyScott Vitter. Document listing for queries with excluded pattern. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching*, volume 7354 of *Lecture Notes in Computer Science*, pages 185–195. Springer Berlin Heidelberg, 2012.

17 Wing-Kai Hon, Sharma V. Thankachan, Rahul Shah, and Jeffrey Scott Vitter. Faster compressed top-k document retrieval. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013*, pages 341–350. IEEE, 2013.

18 KasperGreen Larsen, J.Ian Munro, JesperSindahl Nielsen, and SharmaV. Thankachan. On hardness of several string indexing problems. In AlexanderS. Kulikov, SergeiO. Kuznetsov, and Pavel Pevzner, editors, *Combinatorial Pattern Matching*, volume 8486 of *Lecture Notes in Computer Science*, pages 242–251. Springer International Publishing, 2014.

19 Yossi Matias, S. Muthukrishnan, SüleymanCenk Sahinalp, and Jacob Ziv. Augmenting suffix trees, with applications. In Gianfranco Bilardi, GiuseppeF. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms — ESA' 98*, volume 1461 of *Lecture Notes in Computer Science*, pages 67–78. Springer Berlin Heidelberg, 1998.

20 J. Ian Munro. Tables. In Vijay Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science, 16th Conference, Hyderabad, India, December 18-20, 1996, Proceedings*, volume 1180 of *Lecture Notes in Computer Science*, pages 37–42. Springer, 1996.

21 J. Ian Munro, Gonzalo Navarro, Jesper Sindahl Nielsen, Rahul Shah, and Sharma V. Thankachan. Top- k term-proximity in succinct space. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation – 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2014.

22 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 657–666. ACM/SIAM, 2002.

23 Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1066–1077. SIAM, 2012.

24 Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1066–1077. SIAM, 2012.

25 Gonzalo Navarro and Sharma V. Thankachan. Faster top-k document retrieval in optimal space. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *String Processing and Information Retrieval – 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, volume 8214 of *Lecture Notes in Computer Science*, pages 255–262. Springer, 2013.

26 Manish Patil, Sharma V Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 266–277. ACM, 2014.

27 Kunihiko Sadakane and Gonzalo Navarro. Fully-functional succinct trees. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 134–149. SIAM, 2010.

A Range Transformation using Fractional Cascading

We employ the fractional cascading idea of Chazelle et.al [3] for predecessor searching in CS array. Successor searching and CD array are handled in a similar way. The idea is to merge the CS array for siblings and propagate the predecessor information from bottom-to-top. Two arrays are used for this purpose: merged siblings array MS and merged children array MC. Let h_i be an internal node in T_H^t having sibling h_j and two children leaf nodes h_u and h_v. Array MC_u (resp. MC_v) is same as CS_u (resp. CS_v) and stored in h_u (resp. h_v). The arrays CS_u and CS_v are merged to form a sorted list MS_{uv}. Note that, CS_v values are strictly greater than CS_u; therefore, CS_u and CS_v form two disjoint partitions in MS_{lr} after sorting. We denote the left partition as MS^l_{uv} and the right partition as MS^r_{uv}. We also store a pointer from each value in MS^l_{uv} (MS^r_{uv}) to its corresponding value in MC_u (resp. MC_v). The list MC_i is formed by merging CS_i with every second item from MS_{lr}. With each item x in MC_i, we store three numbers: the predecessor of x in CS_i, the predecessor of x in MS^l_{uv} and the predecessor of x in MS^r_{uv}. Total space required is linear in the number of chains, and is bounded by $O(n)$ words.

Using this data structure, we show how to find predecessor efficiently. Let h_w be an ancestor node of h_z in T_H^t. We want to traverse h_w to h_z path and search for the predecessor of x in CS_i, where h_i is a node on the h_w to h_z path. When we traverse from a parent node h_i to a child node h_j, at first we obtain the predecessor value in parent node using MC_i. If h_j is the left (resp. right) children of h_i, we obtain the predecessor value in MS^l_{jk} (resp. MS^r_{jk}), where h_k is the sibling of h_j. Following the pointer stored at MS^l_{jk} or MS^r_{jk}, we can get the predecessor value at MC_j, and proceed the search to the next level. This way we can obtain the transformed range at each level in $O(1)$ time.

On Density, Threshold and Emptiness Queries for Intervals in the Streaming Model

Arijit Bishnu[1], Amit Chakrabarti[2], Subhas C. Nandy[1], and Sandeep Sen[3]

1 **Indian Statistical Institute, Kolkata, India**
{arijit,nandysc}@isical.ac.in
2 **Department of Computer Science, Dartmouth College, Hanover, USA**
ac@cs.dartmouth.edu
3 **Department of Computer Science Engineering, Indian Institute of Technology – Delhi, New Delhi, India**
ssen@cse.iitd.ernet.in

Abstract

In this paper, we study the maximum density, threshold and emptiness queries for intervals in the streaming model. The input is a stream $\mathcal{S}$ of n points in the real line $\mathbb{R}$ and a floating closed interval W of width α. The specific problems we consider in this paper are as follows.

- Maximum density: find a placement of W in $\mathbb{R}$ containing the maximum number of points of $\mathcal{S}$.
- Threshold query: find a placement of W in $\mathbb{R}$, if it exists, that contains at least Δ elements of $\mathcal{S}$.
- Emptiness query: find, if possible, a placement of W within the extent of $\mathcal{S}$ so that the interior of W does not contain any element of $\mathcal{S}$.

The stream $\mathcal{S}$, being huge, does not fit into main memory and can be read sequentially at most a constant number of times, usually once. The problems studied here in the geometric setting have relations to frequency estimation and heavy hitter identification in a stream of data. We provide lower bounds and results on trade-off between extra space and quality of solution. We also discuss generalizations for the higher dimensional variants for a few cases.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Density, threshold, emptiness queries, interval queries, streaming model, heavy hitter, frequency estimation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.336

1 Introduction

Motivated by problems related to chip density and thermal analysis in VLSI [4, 19], researchers in computational geometry have looked at problems involving windowing queries on a point set [5], such as maximum empty rectangle query [2] and maximum density query [23]. Windowing queries or their one-dimensional counterpart – interval queries – have motivations in geospatial and sensor network applications [16], where huge amounts of data are generated continuously in a stream, and communication is very expensive. Thus, it is preferable to communicate an appropriate summary of the data. Moreover, devices used for this purpose have limited memory. This calls for solving the problems on streaming data using limited amount of memory.

We consider density, threshold, and emptiness queries for a fixed-length interval among points in $\mathbb{R}$ in the streaming model [3, 22]. In the *pure* or *one-pass streaming model*, the data

can be read only once and in the *multi-pass streaming model*, the data can be read more than once but always in the same order; in both cases the data is read-only. Apart from the number of passes, the other crucial issues in the streaming model are the amount of working memory used to process the input and the time taken, per item in the stream, to update this working memory. Ideally, both these quantities should preferably be significantly sub-linear in the size of the input.

1.1 The Computational Model and Problems Considered

Our input is a parameter $\alpha > 0$, representing the width of a *closed* interval W, plus a stream (i.e., sequence) $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$ of n points, where each $s_i \in \mathbb{R}$. We consider certain interval queries that involve placing W suitably among the points in $\mathcal{S}$ so as to satisfy certain objectives. The combinatorial nature of these queries ensures that it suffices to consider only those placements where the left end-point of W coincides with a point in $\mathcal{S}$. Therefore, we define, for each $s \in \mathcal{S}$, the set $\mathcal{I}_s(\alpha) := \{x' \mid x' \in \mathcal{S} \text{ and } s \leqslant x' \leqslant s + \alpha\}$.

Our problems of interest are as follows.

- The *maximum density problem*, where the goal is to find $\max_{s \in \mathcal{S}} |\mathcal{I}_s(\alpha)|$ and a choice of s that achieves the maximum; the problem is denoted as MAX-DENSE.
- The *sorted-order* maximum density problem, denoted MAX-DENSE-SORTED, where the stream $\mathcal{S}$ arrives in a sorted fashion, i.e., $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n$ and the goal is the same as above.
- The *threshold query* problem, denoted THRESHOLD, where we must report whether there exists an s with $|\mathcal{I}_s(\alpha)| \geqslant \Delta$; the parameter Δ is known in advance.
- The *emptiness query* problem, denoted EMPTINESS, where we must report whether there exists an $s \in \mathcal{S} \setminus \{\max \mathcal{S}\}$ with $|\mathcal{I}_s(\alpha)| = 1$. This amounts to asking whether the *open* interval $\text{int}(W)$ can be placed within $[\min \mathcal{S}, \max \mathcal{S}]$ so that it avoids all points in $\mathcal{S}$.

Computations take place in a word RAM with word size large enough to hold the parameters α and Δ, a single point s_i, and a $\lceil \log n \rceil$-bit counter. This essentially means that all "real numbers" appearing as inputs are in fact rationals with bounded bit precision. In our algorithms, the precision of intermediate computations will be within a constant factor of the word size. The stream length n may not be known *a priori*, but thanks to standard techniques we can often nevertheless design algorithms pretending that it is.

1.2 Our Results

We first observe that MAX-DENSE is hard: even a randomized algorithm that approximates $OPT := \max_s |\mathcal{I}_s(\alpha)|$ up to a large constant factor requires $\Omega(n)$ bits of space.[1] On the other hand, MAX-DENSE-SORTED admits a deterministic $(1-\epsilon)$-factor approximation[2] using $O(\epsilon^{-1} \log(\epsilon n))$ words of working space. Returning to MAX-DENSE, we show that given a Δ such that $OPT \geqslant \Delta$, we can compute a $(1-\epsilon)$-factor approximation with high probability using $O(\frac{n \log n}{\epsilon^3 \Delta})$ words. We also suggest a 3-factor approximation algorithm that runs in $O(n \log n)$ time. For the natural generalization of MAX-DENSE to $\mathbb{R}^2$, we show that a 6-approximation can be obtained in $O(n \log M)$ time using $O(M)$ words of space where M is the size of the maximum independent set of the copies of W positioned such that a specific (say

[1] As is common in streaming algorithms, lower bounds are expressed in bits and upper bounds in words.
[2] A γ-factor approximation algorithm means a placement of W at a point $s \in \mathcal{S}$, such that $OPT \geqslant |\mathcal{I}_s(\alpha)| \geqslant \gamma \cdot OPT$.

top-left) corner is at each point in $\mathcal{S}$. For THRESHOLD, we provide approximate deterministic as well as random sampling based algorithms, achieving space bounds of $O_\epsilon(n/\Delta)$[3]. We prove a randomized $\Omega(n/\Delta^2)$ lower bound, showing that this is nearly tight. For EMPTINESS, we prove a strong lower bound of $\Omega(n)$ in the randomized case and an especially strong lower bound of $\frac{1}{2}n - O(1)$ in the deterministic case.

1.3 Related Work

The need to process massive data has generated considerable interest in the data streaming model [15, 22]. The geometric problems we study here have close ties with frequency moment estimation [3] and heavy hitter identification [11]; see also [25] for a short summary. In the extensively-studied frequency moments problem, the stream $\mathcal{S}$ consists of n integers, each in $\mathcal{M} := \{1, 2, \ldots, m\}$ for some $m = n^{\Theta(1)}$. Let $f_i = |\{j \mid s_j = i\}|$ denote the frequency of i in $\mathcal{S}$, and define for each $k \geqslant 0$, $F_k := \sum_{i=1}^{m} f_i^k$. By convention, F_0 is the number of distinct elements and F_∞ is $\max_{i \in \mathcal{M}} f_i$. Up to logarithmic (in n and m) factors, the space complexity for approximating F_k is known to be $\Theta(1)$ for $0 \leqslant k \leqslant 2$ and $\Theta(n^{1-2/k})$ for $k > 2$ [3, 17, 7]. In particular, approximating F_∞ to a large constant factor, e.g., 100, requires $\Theta(m)$ space [26].

In frequency estimation, apart from the sequence $\mathcal{S}$, we have a threshold κ, $0 < \kappa < 1$ and we have to maintain an estimate $\bar{f}_i$, of f_i, such that $f_i - \kappa n \leqslant \bar{f}_i \leqslant f_i$. In heavy hitter identification, we have two parameters ξ and κ, $0 < \xi < \kappa < 1$, and we need to output all elements whose frequency is more than κn, and no element whose frequency is less than $(\kappa - \xi)n$ should be reported. Starting with the result of Misra and Gries [21], there have been a host of results [10, 12].

In the context of the present problem, Emek et al. [13] proposed a 2-factor approximation algorithm for computing the maximum independent set of intervals in streaming setup, that uses space linear in the size of the output. They also proposed a matching lower bound claiming that an approximation ratio of $2-\epsilon$ cannot be obtained by any randomized streaming algorithm with space significantly smaller than the size of the input (much larger than the output size). Recently, Cabello and Pérez-Lantero [6] showed that if the end-points of the intervals are in the set $\{1, 2, \ldots, n\}$, then an estimate $\hat{M}$ of the maximum independent set M can be obtained in space polynomial in ϵ^{-1} and $\log n$ which satisfies $\frac{1}{2}(1-\epsilon)M \leqslant \hat{M} \leqslant M$ with probability at least $\frac{2}{3}$. For equal length intervals, the estimate $\hat{M}$ of M satisfying $\frac{2}{3}(1-\epsilon)M \leqslant \hat{M} \leqslant M$ can be obtained using same amount of space satisfying the same probability bound.

In the multipass streaming model, there have been results related to approximate convex hull [16], approximate minimum enclosing ball [9]. Agarwal et al. [1] proposed a general technique for approximating various extent measures of a point set P in $\mathbb{R}^d$ in the streaming model. Chan [8] raised the issue of getting $O(1)$ space streaming algorithms for these problems. Apart from these, Har-Peled and Mazumdar [14] gave a $(1+\epsilon)$-approximation of the k-median and k-mean clustering of a stream of points in $\mathbb{R}^d$.

2 Interval Placement for Maximum Density

We start by observing that MAX-DENSE generalizes the problem of frequency moment estimation. Given a stream $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$ of integers, we can consider its elements as points in $\mathbb{R}$. Take $\alpha = 0$. Then, for all $s \in \mathcal{S}$, the cardinality $|\mathcal{I}_s(\alpha)|$ is simply the frequency

[3] $O_\epsilon(f)$ denotes ϵ in the expression of f is treated as a constant.

of s. Therefore $\max_s |\mathcal{I}_s(\alpha)| = F_\infty(\mathcal{S})$. For the F_∞ problem, we recall an important result observed (though not formally written as a theorem) in Alon et al. [3]. Based on communication lower bounds for the multi-party SET-DISJOINTNESS problem, it follows that distinguishing $F_\infty(\mathcal{S}) = 1$ from $F_\infty(\mathcal{S}) \geqslant \Delta$ requires $\Omega(n/\Delta^2)$ space. This holds even for randomized and multipass streaming algorithms using $O(1)$ passes.

Based on the above, we obtain the following strong lower bound.

▶ **Theorem 2.1.** *For the* MAX-DENSE *problem, put* $OPT := \max_s |\mathcal{I}_s(\alpha)|$. *Any randomized constant-pass algorithm that distinguishes* $OPT \geqslant \Delta$ *from* $OPT = 1$ *with probability* $\geqslant 2/3$ *uses* $\Omega(n/\Delta^2)$ *bits of space. In particular, approximating* OPT *to any constant fraction requires* $\Omega(n)$ *space.*

If one wants to avoid degeneracy, a simple perturbation argument allows one to replace the hard instance for MAX-DENSE implicit above with one in which all the points in the stream are distinct and $\alpha > 0$.

2.1 Points in Sorted Order: The Problem max-dense-sorted

In light of Theorem 2.1, we consider the easier variant MAX-DENSE-SORTED, where the input stream satisfies $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n$. For this variant, we start by describing an output sensitive procedure, followed by a 2-approximation algorithm and finally, we give a very space-efficient deterministic algorithm achieving a $(1+\epsilon)$-approximation.

The optimum solution OPT can be computed using OPT counters using the following simple *output sensitive* procedure. We allocate counters to count points in $\mathcal{I}_{s_1}(\alpha), \mathcal{I}_{s_2}(\alpha), \ldots$, until we get a point $p \in \mathcal{S}$ that lies outside $s_1 + \alpha$. At this point of time, the counter for $\mathcal{I}_{s_1}(\alpha)$ is closed, and a new counter for $\mathcal{I}_p(\alpha)$ is initiated. At any instant of time, the maximum of the contents of all closed counters is stored in OPT. At the end of the stream, all the active counters are closed, and we report the content of OPT. Thus the maximum number of counters active at any point of time is bounded by OPT. For n points uniformly distributed in the interval $[\ell, u]$, this algorithm will be very space efficient when $\frac{u-\ell}{\alpha} \geqslant n/polylog(n)$ – see Lemma 1.1 of the Appendix.

A simple 2-approximation algorithm is easy to obtain. (We would need this idea when we discuss THRESHOLD.) We initiate a counter for counting points in $\mathcal{I}_{s_1}(\alpha)$. The counting continues until we get a point $p \in \mathcal{S}$ that lies outside $[s_1, s_1 + \alpha]$. The same counter now starts counting for $\mathcal{I}_p(\alpha)$. Finally, report the interval with maximum number of points. The approximation ratio follows from the fact that the optimal α-interval spans on two consecutive α-intervals $\mathcal{I}_{s_i}(\alpha)$ and $\mathcal{I}_{s_{i+1}}(\alpha)$ for which we have computed the number of points.

Next, we discuss the $(1+\epsilon)$-approximation, the main result of this subsection. For our algorithm, we introduce a subroutine that we call *the* (k, B)*-process* (k and B are positive integers), defined as follows. We maintain up to B active counters for certain sets $\mathcal{I}_s(\alpha)$, plus a register $n_{\max}$ to record the maximum value ever seen in a counter. At every kth stream element s – i.e., for $s \in \langle s_1, s_{1+k}, s_{1+2k}, \ldots \rangle$ – first close all active counters for sets $\mathcal{I}_{s'}(\alpha)$ such that $s > s' + \alpha$, updating $n_{\max}$ as needed. Reclaim the space allocated to all closed counters. Then, if there is space, open a new counter to accurately count $\mathcal{I}_s(\alpha)$; if there is no room – i.e., we are about to open a $(B+1)$th counter – then abort the process instead. Increment all active counters. At the end of the stream, if we haven't aborted, output $n_{\max}$.

We make use of the simple observation that if the (k, B)-process aborts, then $OPT > kB$; otherwise, $OPT - k \leqslant n_{\max} \leqslant OPT$.[4]

[4] The initial idea of classifying OPT was conveyed to one of the authors by Sai Praneeth.

Algorithm 1: active-process (k, B)

Input stream $s_1, s_2 \ldots$
A set $\mathcal{C}$ of B counters. Initialize the first counter for $\mathcal{I}_{s_1}(\alpha)$. $n_{\max} \leftarrow 0$;
for $s \in \{s_{1+k}, s_{1+2k}, s_{1+ik} \ldots\}$ **do**
  Close all active counters for $\mathcal{I}_{s'}(\alpha)$ for $s > s' + \alpha$ and update $n_{\max}$;
  Increment all active counters (for $\mathcal{I}_{s'}(\alpha)$ for $s \leqslant s' + \alpha$) ;
  if $|\mathcal{C}| < B$ **then**
    initialize a new counter for $\mathcal{I}_s(\alpha)$
  else
    abort current process
Return $(n_{\max})$

Using these processes, we design our algorithm as follows. Let ℓ be an integer to be chosen later. Choose $k = \lfloor \epsilon n \rfloor^{1/\ell}$ and $B = \lceil k/\epsilon \rceil$. For $i = 0$ to ℓ, in parallel, run the (k^i, B)-process. At the end of the stream, output $n_{\max}$ corresponding to the smallest i such that the (k^i, B)-process did not abort. Let n^* be this output. There will always exist a suitable i because, as can be checked easily, the (k^ℓ, B)-process cannot abort.

▶ **Claim 2.2.** *We have* $(1 - \epsilon)OPT \leqslant n^* \leqslant OPT$.

Proof. The upper bound on n^* follows directly from the observation we recorded. For the lower bound, first suppose that the $(1, B)$-process did not abort. Then that process accurately counted $\mathcal{I}_s(\alpha)$ for *every* s in the stream, so $n^* = OPT$. Next, suppose that the (k^{i-1}, B)-process aborted, where $i > 0$. By our observation, $OPT > k^{i-1}B$. Also, because the (k^i, B)-process did not abort, by the other part of our observation, we have $n^* \geqslant OPT - k^i = OPT - (k^{i-1}B)(k/B) > (1 - k/B)OPT \geqslant (1 - \epsilon)OPT$. ◀

The previous algorithm uses at most B counters and one extra register in each of its $\ell + 1$ parallel processes. Therefore, its space usage is $O(\ell B) = O(\ell \epsilon^{-1}(\epsilon n)^{1/\ell})$ words. We optimize this by setting $\ell \approx \log(\epsilon n)$.

▶ **Theorem 2.3.** *For all* $\epsilon \in (0, 1)$*, there is a deterministic* $(1 - \epsilon)$*-factor approximation for* MAX-DENSE-SORTED*, using* $O(\epsilon^{-1} \log(\epsilon n))$ *words of space.*

2.2 Points in Arbitrary Order: The Problem max-dense

We return to MAX-DENSE, with points arriving in an arbitrary (unsorted) order. Here, we assume that no two points in $\mathcal{S}$ have the same x-coordinate. For some appropriate constant c, we sample independently every element from the stream with probability $p = cn_k \log n/n$, where n_k is the size of a k-sample (for some appropriate k depending on the application). Let R' denote this sample. We sort the elements in R' and then choose a subset $R \subset R'$ by selecting every $c \log n$-th element from the sorted sequence of R'.

▶ **Claim 2.4.** *For a given* ϵ $(0 < \epsilon < 1)$*, there exists some appropriate* c *such that for every pair of consecutive elements* $r_i, r_{i+1} \in R$*, we have*

$$\Pr[k(1 - \epsilon) \leqslant |\mathcal{S} \cap [r_i, r_{i+1}]| \leqslant k(1 + \epsilon)] \geqslant 1 - 1/n.$$

Proof. For any consecutive (sorted) sample points r_i, r_{i+1}, if the number of unsampled elements, U_i is less than $k(1 - \epsilon)$ elements, it implies that more than $c \log n$ elements were chosen from U_i. Every element is sampled independently with probability $p = \frac{c \log n}{k}$

$(n_k = n/k)$, so the expected number of samples in U_i is $c(1-\epsilon)\log n$. Let X_i be a random variable representing the number of samples from U_i. From Chernoff bounds, we have $\Pr[X_i \geqslant (1+\delta)\mathbb{E}[X_i]) \leqslant \exp(-\delta^2\mathbb{E}[U_i]/2)$ where $\mathbb{E}[U_i] = c(1-\epsilon)\log n$ and $1+\delta = 1/(1-\epsilon)$, i.e., $\delta \approx \epsilon$. For an appropriately large value of $c = \Omega(\frac{1}{\epsilon^2})$, we can bound this probability by $\frac{1}{n^2}$. The above calculation holds for a pair of sample points that are consecutive, but we can easily uncondition it by multiplying with the probability that they are consecutive (which is less than 1). Therefore, none of the intervals contain less than $k(1-\epsilon)$ points.

A similar calculation yields an upper bound on the number of unsampled elements in an interval. ◀

The above proof says that the sample R can be treated as a k-sample of $\mathcal{S}$ for getting an approximate solution for MAX-DENSE. If $OPT \geqslant \Delta$, then we choose $k = \lfloor \epsilon\Delta \rfloor$ to have the size of the k-sample as $n_k = n/(\epsilon\Delta)$. If OPT_R is the maximum count of an α-interval corresponding to an element of R, then with high probability we have $(1-\epsilon)^2 OPT \leqslant k(OPT_R - 1) \leqslant (1+\epsilon)^2 OPT$, where the extra $1 \pm \epsilon$ factor is due to the sampling variance as per the previous claim. Substituting $\epsilon = \epsilon/2$, we have an $(1-\epsilon)$ approximation.

▶ **Theorem 2.5.** *If $OPT \geqslant \Delta$, then for any $0 < \epsilon < 1$, OPT can be approximated within a factor $(1-\epsilon)$ with high probability using $O\left((\epsilon\Delta)^{-1} cn \log n\right)$ space where $c = O(\epsilon^{-2})$. The value Δ is an input to the streaming algorithm.*

▶ Remark. There is a huge gap between the space requirements of MAX-DENSE and MAX-DENSE-SORTED.

An output sensitive algorithm

With the strong lower bound already shown in Theorem 2.1, our goal is to have an output sensitive algorithm. Here, the intervals (of unequal length) are created online, and stored in a height balanced binary tree $\mathcal{T}$. We use the term *short*, *exact* and *long* to denote the intervals having length less than or equal to or greater than α. With each created interval I, we store its *span* $\delta(I)$ and *count* fields $count(I)$. Every interval J having span $\delta(J) > \alpha$ has $count(J) = 0$. Initially, a single interval $(-\infty, \infty)$ is present in $\mathcal{T}$. When a point p arrives, the tree $\mathcal{T}$ is searched to identify the interval (say $I = [a, b]$) containing p. If $\delta(I) \leqslant \alpha$, $count(I)$ is incremented. If $\delta(I) > \alpha$, we insert an interval J with one end point at p and of span $\delta(J) = \alpha$ in $\mathcal{T}$. Here the following cases need to be considered:

Case 1: $J = [p, q]$ is contained in an existing interval $I = [a, b]$ with $\delta(I) > \alpha$ (see Figure 1(a)). We delete I from $\mathcal{T}$ and insert three intervals $I_1 = [a, p]$, $I_2 = J = [p, q]$ and $I_3 = [q, b]$ in $\mathcal{T}$ with $count(I_1) = count(I_3) = 0$ and $count(I_2) = 1$. Here I_2 is an *exact* interval. I_1 and I_3 may be of any type.

Case 2: If $J = [p, q]$ overlaps with the some other interval $I' = [c, d]$ $(\neq I)$ then I' must be an α-interval. We consider $J' = [q', p]$ of length α, with p as its right end-point (where q' is not an input point).

Case 2.1: If J' does not overlap with any other interval (See Figure 1(b)), then we delete I and insert three intervals $I_1 = [a, q], I_2 = J' = [q, p], I_3 = [p, b]$ in $\mathcal{T}$. Here I_2 is *exact* and I_3 is *short*. I_1 may be of any type.

Case 2.2: If $J' = [q, p]$ overlaps with an interval $I'' = [e, f]$ (See Figure 1(c)), then I'' is also of length α, and we have $I = [a, b] = [f, c]$ with $\alpha < \delta(I) \leqslant 2\alpha$. Here I is replaced with an *exact* interval $I_1 = [a, a+\alpha]$ and a *short* interval $I_2 = [a+\alpha, b]$ in $\mathcal{T}$ with $count[I_1] = 1$ and $count[I_2] = 0$.

The intervals created are characterized as follows.

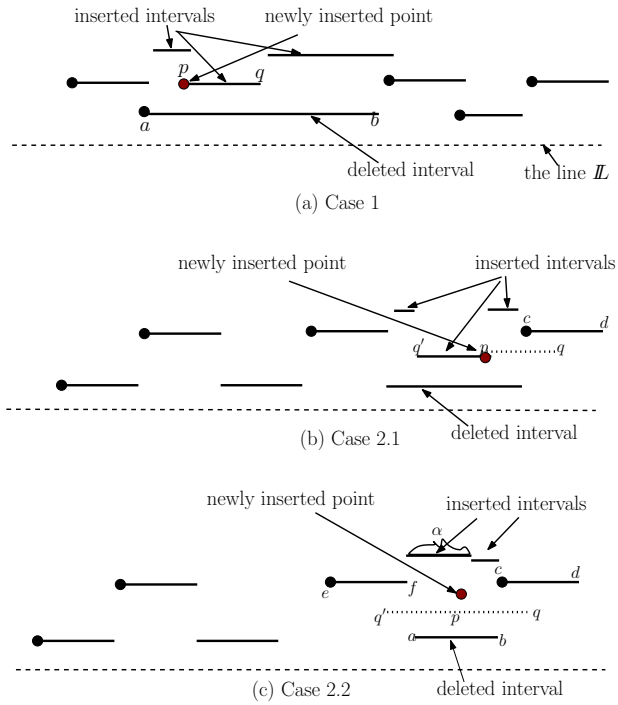


Figure 1 Processing of a new point in the stream: Here the dotted line is $\mathbb{L}$, the existing intervals, and the intervals to be inserted for a new stream element p, are shown.

▶ **Lemma 2.6.** *(a) At any point of time during the execution, the two adjacent intervals of any* short *interval are* exact *intervals.*

(b) The interval with maximum frequency contains at least $\frac{1}{3}OPT$*, where* OPT *is the maximum number of points of* S *that an* α*-interval can contain.*

Proof. Part (a) follows from the fact that a *short* interval is created by splitting a long interval of length less than 2α (see Case 2.2 earlier).

Part (b) follows from the fact that the interval corresponding to the OPT may span at most 3 intervals of $\mathcal{T}$. ◀

We now give an estimate of the size of the work-space for maintaining $\mathcal{T}$ using the size of maximum independent set (MIS) of the set of α-intervals anchored at each point of $\mathcal{S}$.

▶ **Lemma 2.7.** *If* χ *is the number of* exact *intervals in* $\mathcal{T}$*, then* $\frac{2}{3}MIS \leqslant \chi \leqslant MIS$.

Proof. The right-hand side of the inequality trivially follows. We need to prove the left-hand side of the inequality. Note that, the *long* intervals do not contain any point. Also, in the optimum solution, it is not possible to have more than one interval generated by two points inside a *short* or *exact* interval since its length is less than or equal to α. Again, both the neighbors of a *short* interval in $\mathcal{T}$ are *exact* intervals (by Lemma 2.6(a)). Thus, we have the left hand side of the inequality, since in the worst case, there may exist an instance where each triplet (*exact, short, exact*) of intervals is separated by a *long* interval. ◀

The output sensitive algorithm is summed up in the next Theorem. Note that MIS in the worst case can be linear.

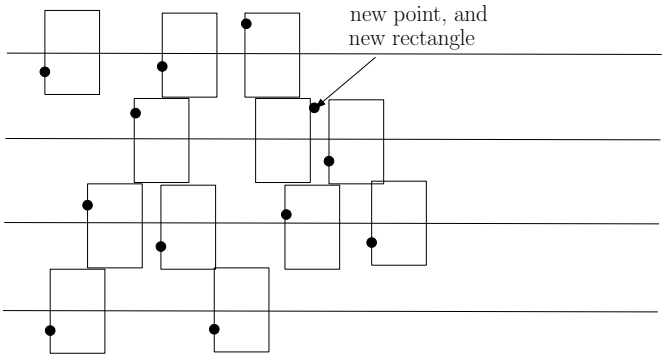


Figure 2 Insertion of a point.

▶ **Theorem 2.8.** *For* MAX-DENSE*, an α-interval $\hat{I}$ can be computed in $O(n \log n)$ time using $O(MIS)$ extra work-space, such that the number of points of S that $\hat{I}$ covers is at least $\frac{1}{3}OPT$, where OPT is the maximum number of points of S that can lie inside an α-interval, and MIS is the size of the independent set among all α-intervals with left end-points anchored at the points in S.*

Proof. The approximation factor follows from Lemma 2.6(b). The time complexity follows from the fact that we are spending $O(1)$ time for processing each point. The space complexity follows from the number of intervals stored in $\mathcal{T}$. By Lemma 2.6(a), the number of short intervals is less than the number of *exact* intervals. The number of *large* intervals can be at most 1 more than the number of *exact* intervals in the worst case, and Lemma 2.7 says that the number of exact intervals is less than or equal to MIS. Thus, the number of intervals in $\mathcal{T}$ is $O(MIS)$. ◀

2.3 Maximum Density with Points in Two Dimensions

Here a stream of points $\mathcal{S}$ is arriving online in $\mathbb{R}^2$, and a rectangular window W of size $\alpha \times \beta$ is given. The objective is to report the position of W that contains maximum number of points of $\mathcal{S}$. We can formulate the problem as follows.

▶ **Definition 2.9.** For each point p in the stream $\mathcal{S}$, an *exact copy of* W is a rectangle of size $\alpha \times \beta$ with p on its top-left corner.

Thus, our objective is to compute the largest clique in the intersection graph of these exact copies of W, where the bottom-right corner of W is to be placed to contain the maximum number of points. In order to handle this streaming version of the problem, we create copies of W in a slightly different manner, and show that the approximation factor of our proposed algorithm is 6.

As in MAX-DENSE, here also we create a covering of points of $\mathcal{S}$ in $\mathbb{R}^2$ with disjoint rectangles of size $\alpha' \times \beta$, $\alpha' \leqslant \alpha$, such that each rectangle contains at least one point of $\mathcal{S}$. As the points in $\mathcal{S}$ arrive, we create these rectangles online (see Figure 2), and store them in a data structure. When a point $p \in \mathcal{S}$ arrives, if it lies inside an existing rectangle then the *count* of that rectangle is increased by one; otherwise, a new rectangle is created that contains p, and its count is set to 1. When the stream ends, the rectangle having the maximum *count* is reported.

We assume that the points in $\mathcal{S}$ have positive x and y coordinates. We conceptually split the floor using horizontal lines $y = 0, \beta, 2\beta, \ldots$. On arrival of a point $p = (p_x, p_y) \in \mathcal{S}$, if it is

not inside any one of the existing rectangles, we create a new rectangle as follows: we compute $i = \frac{p_y}{\beta}$. The vertical span of the created rectangle is $[i\beta - \frac{\beta}{2}, i\beta + \frac{\beta}{2}]$ or $[(i+1)\beta - \frac{\beta}{2}, (i+1)\beta + \frac{\beta}{2}]$ depending on whether $p_y - i\beta < \frac{\beta}{2}$ or $p_y - i\beta > \frac{\beta}{2}$. As in MAX-DENSE, the horizontal span of this rectangle is decided such that it must contain p, its horizontal width is at most α, and it does not overlap on any other existing rectangles in the data structure.

We store the horizontal lines containing at least one rectangle in a height-balanced binary tree $\mathbb{T}$. The rectangles having vertical span $[i\beta - \frac{\beta}{2}, i\beta + \frac{\beta}{2}]$ are stored in the form of disjoint intervals on the horizontal line $y = i\beta$ in a height-balanced binary tree $\mathcal{T}_i$ as in MAX-DENSE, and is attached with the i-th node of $\mathbb{T}$. The following theorem generalizes the result.

▶ **Theorem 2.10.** *Given a stream $\mathcal{S}$ of n points in $\mathbb{R}^2$, executing a single pass over the stream, one can compute a position of placing a rectangular window W of a given size in $\mathbb{R}^2$ such that it encloses at least $\frac{OPT}{6}$ points, where OPT is the maximum number of points in $\mathcal{S}$ that can be enclosed by placing the window W.*

The time and work-space required for executing this algorithm is $O(n \log R_{opt})$ and $O(R_{opt})$ respectively, where R_{opt} is the size of the maximum independent set of the exact copies of W *corresponding to the points in $\mathcal{S}$.*

Proof. Let W_{opt} be the optimum position of the rectangle W, and OPT be the number of points in W_{opt}. Our algorithm has reported W_{max} that contains maximum number of points with respect to our definition of rectangles for covering the points in the plane. Observe that W_{opt} can overlap on at most 6 different rectangles according to our layout (see Figure 2); one of these rectangles must contain at least $\frac{1}{6}OPT$ number of points. Thus if OPT be the number of points in W_{opt}, then W_{max} contains at least $\frac{1}{6}OPT$ points.

If M is the number of rectangles present in the data structure $\mathbb{T}$, then, processing each point takes $O(\log M)$ time in the worst case. Thus, the time complexity of the algorithm is $O(n \log M)$ time, and it uses $O(M)$ extra work-space.

Now, we show that $M \leqslant 2R_{opt}$. Consider the *exact copy of* W corresponding to a point $p \in \mathcal{S}$ (see Definition 2.9). If $i = \lfloor \frac{p_y}{\beta} \rfloor$, then assign p (and hence, W) to both the lines $y = i\beta$ and $y = (i+1)\beta$. Now, consider each horizontal line separately, and consider the intersection graph of the intervals of width α corresponding to the assigned points with this line. If I_i is the maximum independent set of this interval graph, and M_i is the set of intervals stored in $\mathcal{T}_i$, then $|M_i| \leqslant |I_i|$ (by Lemma 2.7). Thus, $M = \sum_{i=1}^{k} |M_i| \leqslant \sum_{i=1}^{k} |I_i|$, where k is the number of horizontal splitting lines of the floor. Again, since the exact copy of W corresponding to each point $p \in \mathcal{S}$ is assigned to two adjacent splitting lines, the two sets $I_{odd} = \cup_{i=1,3,\ldots,k} I_i$ and $I_{even} = \cup_{i=2,4,\ldots,k} I_i$ are independent, and the size of each of them is less than or equal to R_{opt}. Thus, we have the desired result $M = \sum_{i=1}^{k} |M_i| \leqslant \sum_{i=1}^{k} |I_i| \leqslant I_{odd} + I_{even} \leqslant 2R_{opt}$. ◀

3 Threshold and Emptiness Queries

Now we turn to the other types of interval queries, namely threshold and emptiness queries respectively.

3.1 Threshold Queries: The Problem threshold

Recall that the goal of THRESHOLD is that given prespecified α and Δ, to determine whether an α-interval can be placed to contain at least Δ of the points in the input stream $\mathcal{S}$. As already noted, this is equivalent to finding whether there exists an $s \in \mathcal{S}$ such that $|\mathcal{I}_s(\alpha)| \geqslant \Delta$. We first discuss a two-pass deterministic algorithm for THRESHOLD, followed by a one-pass randomized approximation algorithm.

Algorithm 2: Update(s)

```
begin
    Initialize all ⌈1/ϵ⌉ counters to 0;
    if (s belongs to any of the intervals being tracked by counters in C) then
        increment the counter for the α-interval in L in which s belongs;
    else
        if (|C| < ⌈1/ϵ⌉) then
            Open a new counter, that tracks the number of points inside I_i(α), with a
            count of 1, where i = ⌈s/α⌉;
        else
            Decrement all counters in C by 1 and return to the available pool of
            counters all counters that reach 0;
```

A Deterministic Algorithm

The idea is to use the Misra-Gries summary [21] which is basically a generalization of the classical majority finding algorithm. Subsequent researchers [12, 18, 25] have used this idea for frequency estimation and finding heavy hitters. Apart from the sequence $\mathcal{S}$, we have a threshold ϵ, $0 < \epsilon < 1$, and we can maintain an estimate $\bar{f}_i$, of $f_i = |I_{s_i}(\alpha)|$, which is the frequency of $s_i \in \mathcal{S}$, such that $f_i - \epsilon n \leqslant \bar{f}_i \leqslant f_i$, and $\bar{f}_i$ for all α-intervals can be computed using $O(1/\epsilon)$ counters.

We reduce our problem to Misra-Gries summary giving labels to the points in $\mathcal{S}$. Each point $s \in \mathcal{S}$ is labeled as $\lceil s/\alpha \rceil$ – this basically classifies each point s into a set of disjoint canonical intervals $\mathcal{L} = \{\mathcal{I}_1(\alpha) = [0, \alpha), \mathcal{I}_2(\alpha) = [\alpha, 2\alpha), \ldots, \}$. Let us denote the set of counters as $\mathcal{C}$, $|\mathcal{C}| \leqslant \lceil 1/\epsilon \rceil$; the counters in $\mathcal{C}$ would maintain the count of points in some of the $\lceil 1/\epsilon \rceil$ intervals of $\mathcal{L}$. We set $\Delta = \epsilon \cdot n$. Note that, at a time at most $\lceil 1/\epsilon \rceil$ α-intervals are active. The procedure is described next.

Let $f_i = |\mathcal{I}_i(\alpha)|$, denote the number of points inside the i-th canonical interval of $\mathcal{L}$. At the end of the stream, if $\bar{f}_i$ be the value of the counter for the i-th canonical interval of $\mathcal{L}$, ($\bar{f}_i = 0$ if it is decremented to 0 during the process), then it is guaranteed that $\bar{f}_i \in [f_i - \Delta, f_i]$ because of the following ideas as described in [25]. The upper bound is trivial. For the lower bound, let $\bar{f}_i \geqslant f_i - \gamma$ where γ is the number of times the counter for an α-interval can be decremented. Recall that $|\mathcal{C}|$ counters are decremented together. As all α-intervals are disjoint and no point is repeated, we have $\lceil 1/\epsilon \rceil \cdot \gamma \leqslant n$. So, $\gamma \leqslant n\epsilon \leqslant \Delta$. Thus we have $f_i - \Delta n \leqslant \bar{f}_i \leqslant f_i$. Note that, the converse is not true, i.e. even if $\bar{f}_i > 0$ for some s_i, f_i may be less than Δ. This necessitates a second pass, where we can verify the actual counts.

The above gives information only about the canonical intervals. Now using the ideas of the simple 2-approximation algorithm in Section 2.1, we can claim the following about the original question of THRESHOLD– if there exists an s with $|\mathcal{I}_s(\alpha)| \geqslant \Delta$, then there also exists a canonical interval with frequency greater than $\Delta/2$. So, if $\bar{f}_i \geqslant \Delta$, then our answer is yes; if all $\bar{f}_i \leqslant \Delta/2$, then our answer is no. If there exists $\bar{f}_i$ such that $\Delta/2 \leqslant \bar{f}_i \leqslant \Delta$, then the only thing we can say about the THRESHOLD question is that there exists s with $|\mathcal{I}_s(\alpha)| \geqslant \Delta/2$.

▶ **Theorem 3.1.** *There exists a two-pass deterministic algorithm for* THRESHOLD *using* $O(1/\epsilon)$ *counters that gives a 2-factor approximate answer, where* $\Delta = \epsilon \cdot n$.

A Randomized Approximation Algorithm

We propose a one pass randomized approximation algorithm for THRESHOLD that returns the correct answer with high probability $(1 - n^{-\Omega(1)})$. We draw a random sample of points from S where each point is chosen with probability $p = \frac{\log n}{\Delta}$ to generate a random sample R. But n is unknown to us since it is an one pass algorithm. Assume, for now that we know n; we would later resolve this problem.

Note that, the expected space needed for storing R is $\frac{n \log n}{\Delta}$. As every element in R is sampled with a probability p, the expected sample size in any α-interval I containing more than Δ points is $R_I = \Omega(\log n)$. Using Chernoff bounds, it can be shown that with high probability $R_I \geqslant c \log n$. Moreover, if all α-intervals contain fewer than $\frac{\Delta}{\beta}$ points, for some $\beta > 1$, then with high probability, no α-interval contains more than $(c - \epsilon) \log n$ points, for some $\epsilon > 0$. If the given space exceeds $\frac{n \log n}{\Delta}$, then the above scheme works in a straight forward manner by first choosing the sample and subsequently finding the largest α-interval of the sample and then verifying if it exceeds $c \log n$. To extend this idea where n is not known a priori, we can use the idea of Manku and Motwani [20] where the sampling rate is decreased as the stream progresses, so that space usage remains bounded. We can summarize as follows.

▶ **Theorem 3.2.** *There is a one pass $O(\frac{n \log n}{\epsilon^2 \cdot \Delta})$ space bounded randomized algorithm that correctly reports an α-interval containing more than Δ points or asserts that no α-interval contains more than $(1 - \epsilon) \cdot \Delta$ points with high probability.*

▶ Remark. Compared to the two-pass algorithm, it uses $\log n$-factor more space. Compared to Theorem 2.5, the bound is better by a factor ϵ since we are only interested in a threshold.

Space Lower Bound for Threshold Queries

We can obtain a lower bound for THRESHOLD by using the same technique as for MAX-DENSE, i.e., reducing from the F_∞ problem. Suppose a stream of integers $\mathcal{S}$ has the property that either $F_\infty(\mathcal{S}) = 1$ or else $F_\infty(\mathcal{S}) \geqslant \Delta$, for some threshold parameter Δ. Taking $\alpha = 0$ we see that the answers to the threshold query in these two cases are "no" and "yes" respectively. We conclude the following lower bound. As before, the implicit hard instances can be made non-degenerate by perturbation.

▶ **Theorem 3.3.** *A randomized constant-pass algorithm that solves the basic decision version of the* THRESHOLD *problem with parameter Δ requires $\Omega(n/\Delta^2)$ space.*

3.2 Emptiness Queries: The Problem emptiness

In the EMPTINESS problem, the objective is to find whether there exists an empty interval of length α within the extent of the set of points in the data stream. As noted earlier, this is equivalent to determining whether there exists $s \in \mathcal{S} \setminus \{\max \mathcal{S}\}$ such that $|\mathcal{I}_s(\alpha)| = 1$. We show that this problem also admits strong lower bounds.

For this, we reduce from DISJ_m, the two-party SET-DISJOINTNESS communication problem on the universe $\mathcal{M} = \{1, 2, \ldots, m\}$. In the communication problem, Alice gets a set $X \subseteq \mathcal{M}$ and Bob gets a set $Y \subseteq \mathcal{M}$. They must decide whether or not $X \cap Y = \varnothing$. This problem has deterministic communication complexity $m + 1$ and randomized communication complexity $\Omega(m)$, see e.g., [24].

The reduction is as follows. Alice converts X into a stream of elements of $\{0\} \cup (\mathcal{M} \setminus X)$ and Bob similarly converts Y to $(\mathcal{M} \setminus Y) \cup \{m + 1\}$. If $X \cap Y = \varnothing$, then the concatenation

of these streams contains every point in $\{0, 1, \ldots, m+1\}$, so it is impossible to find an empty interval of width $\alpha = \frac{3}{2}$. On the other hand, if X and Y contain a common element z, then the combined stream is missing z, so the open interval $(z-1, z+1)$ is empty. It follows that any algorithm that solves EMPTINESS also solve DISJ_m. The stream created has length $n \leqslant 2m$. Therefore, we obtain the following bounds.

▶ **Theorem 3.4.** *Every randomized constant-pass algorithm that solves* EMPTINESS *requires* $\Omega(n)$ *bits of space. Furthermore, every deterministic algorithm that does the same requires at least* $\frac{1}{2}n - O(1)$ *space.*

4 Conclusion

We studied some problems related to density of points inside intervals in the streaming model. We observed that these problems in geometry are generalizations of frequency moments, frequency estimation and heavy hitters problems. We obtained deterministic as well as randomized approximations using bounded amount of extra space. We proved nearly matching lower bounds on space as well. An interesting open problem would be to look at the higher dimensional variants of the above problems apart from tightening the space bounds.

References

1 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, July 2004.

2 A. Aggarwal and S. Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SCG'87, pages 278–290, 1987.

3 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

4 T. Asano, M. Sato, and T. Ohtsuki. Computational geometric algorithms. In *Layout Design and Verification, Advances in CAD for VLSL (Edited by T. Ohtsuki)*, pages 295–347. North Holland, 1986.

5 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd ed. edition, 2008.

6 Sergio Cabello and Pablo Pérez-Lantero. Interval selection in the streaming model. In *WADS'15*, pages 127–139, 2015.

7 Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *PROC18 # CCC*, pages 107–117, 2003.

8 Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom.*, 35(1-2):20–35, 2006.

9 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. In *Proceedings of the 12th International Conference on Algorithms and Data Structures*, WADS'11, pages 195–206, 2011.

10 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

11 Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *VLDB J.*, 19(1):3–20, 2010.

12 Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Algorithms – ESA 2002, 10th Annual European Symposium*, pages 348–360, 2002.

13 Yuval Emek, Magnús M. Halldórsson, and Adi Rosén. Space-constrained interval selection. In *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 302–313, 2012.

14 Sariel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

15 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridar Rajagopalan. Computing on data streams, 1998.

16 John Hershberger and Subhash Suri. Adaptive sampling for geometric problems over data streams. *Comput. Geom. Theory Appl.*, 39(3):191–208, April 2008.

17 Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208, 2005.

18 Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, 2003.

19 Subhashis Majumder and Bhargab B. Bhattacharya. On the density and discrepancy of a 2d point set with applications to thermal analysis of vlsi chips. *Inf. Process. Lett.*, 107(5):177–182, 2008.

20 Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

21 Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

22 S. Muthukrishnan. Data streams: Algorithms and applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'03, pages 413–413, 2003.

23 Subhas C. Nandy and Bhargab B. Bhattacharya. A unified algorithm for finding maximum and minimum point enclosing rectangles and cuboids. *Int. J. on Computers and Mathematics with applications*, 29(8):45–61, 1995.

24 Alexander Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

25 Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Parallel streaming frequency-based aggregates. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'14*, pages 236–245, 2014.

26 David P. Woodruff. Frequency moments. In *Encyclopedia of Database Systems*, pages 1169–1170. Springer, US, 2009.

A Maximum number of points in an interval having uniformly distributed points

Let the stream $\mathcal{S}$ be any arbitrary permutation of n points where the points are uniformly distributed in the range $[\ell, u]$. Let $\frac{u-\ell}{\alpha} = g(n)$ in where $g(n)$ is a function of n. Then we have the following bound on the maximum number of points in any interval of length α.

▶ **Lemma 1.1.** *The maximum number of points in any interval $I \subset [\ell, u]$ is bounded by $\max\{\frac{n}{g(n)}, c \log n / \log\log n\}$ with probability at least $1 - 1/n$ for $g(n) \leqslant n^{1+\epsilon}$ for any $\epsilon > 0$. For $g(n) \geqslant n^{1+\epsilon}$, it is $O(1)$ with probability $\geqslant 1 - 1/n$.*

Proof. Consider a canonical partition of the range $[\ell, u]$ consisting of intervals $[\ell, \ell+\alpha], [\ell+\alpha, \ell+2\alpha] \ldots [\ell+i\alpha, \ell+(i+1)\alpha]$. Let us denote this set of intervals by $\mathcal{C}$ - from our previous assumption, the number of intervals in $\mathcal{C}$ is bounded by $g(n)$.

Suppose the n points are generated as i.i.d. in $[\ell, u]$, viz., each of the n points is independently generated with uniform distribution in $[\ell, u]$. For a fixed interval $I' \in \mathcal{C}$, the probability that point $q_i, 1 \leqslant i \leqslant n$ is in I' is $p = \frac{\alpha}{u-\ell}$. Let U be random variable that represents the number of points in I' that follows a binomial distribution, so $\mathbb{E}[U] = \frac{n}{g(n)}$. Therefore it follows from the following version of Chernoff bounds

$$\Pr[U \geqslant (1+\Delta)\mathbb{E}[U] \leqslant \left[\frac{e^{\Delta}}{(1+\Delta)^{1+\Delta}}\right]^{E[U]} \tag{1}$$

that the number of points in I' is bounded by $c' \log g(n)/\log\log g(n)$ with probability 1 - $1/g(n)^{c'}$ for some appropriate c' using $\Delta = \frac{\ell-u}{\alpha}$, for $g(n) \geqslant en$. Since the total number of intervals is bounded by $g(n)$, a similar bound follows for all intervals in $\mathcal{C}$ using the union bound and by adjusting the value of c'. For any arbitrary α length interval ($\notin \mathcal{C}$), it intersects at most two intervals in $\mathcal{C}$ and so it cannot exceed $2c \log g(n)/\log\log g(n) = \theta(\frac{\log n}{\log\log n})$.

For $g(n) \leqslant n/\log n)$, the bound of $n/g(n)$ holds with high probability using similar calculations.

For $g(n) \geqslant n^{1+\epsilon}$, $E[U] = n^{-\epsilon}$ and choose $\Delta = cn^{\epsilon}$ for some appropriately large constant c. Substituting in Equation 1 yields the required bound $\Pr[U \geqslant \Omega(1)] \leqslant 1/n$. ◀

Clustering on Sliding Windows in Polylogarithmic Space*

Vladimir Braverman[1], Harry Lang[2], Keith Levin[1], and Morteza Monemizadeh[3]

1 **Department of Computer Science**
Johns Hopkins University
Baltimore, MD, US
vova@cs.jhu.edu, klevin@jhu.edu

2 **Department of Mathematics**
Johns Hopkins University
Baltimore, MD, US
hlang8@jhu.edu

3 **Computer Science Institute**
Charles University
Prague, Czech Republic
monemi@iuuk.mff.cuni.cz

Abstract

In PODS 2003, Babcock, Datar, Motwani and O'Callaghan [4] gave the first streaming solution for the k-median problem on sliding windows using $O(\frac{k}{\tau^4}W^{2\tau}\log^2 W)$ space, with a $O(2^{O(1/\tau)})$ approximation factor, where W is the window size and $\tau \in (0, \frac{1}{2})$ is a user-specified parameter. They left as an open question whether it is possible to improve this to polylogarithmic space. Despite much progress on clustering and sliding windows, this question has remained open for more than a decade.

In this paper, we partially answer the main open question posed by Babcock, Datar, Motwani and O'Callaghan. We present an algorithm yielding an exponential improvement in space compared to the previous result given in Babcock, et al. In particular, we give the first polylogarithmic space (α, β)-approximation for metric k-median clustering in the sliding window model, where α and β are constants, under the assumption, also made by Babcock et al., that the optimal k-median cost on any given window is bounded by a polynomial in the window size. We justify this assumption by showing that when the cost is exponential in the window size, no sublinear space approximation is possible. Our main technical contribution is a simple but elegant extension of *smooth functions* as introduced by Braverman and Ostrovsky [9], which allows us to apply well-known techniques for solving problems in the sliding window model to functions that are not smooth, such as the k-median cost.

1998 ACM Subject Classification I.5.3 Clustering

Keywords and phrases Streaming, Clustering, Sliding windows

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.350

* The first author was supported in part by the National Science Foundation under Grant No. 1447639, the Google Faculty Award and DARPA grant N660001-1-2-4014. Material presented here is solely the responsibility of the authors and does not represent the official view of DARPA or the Department of Defense.
The second author was supported in part by the Franco-American Fulbright Commission.
The fourth author was supported in part by the center of excellence CE-ITI (project P202/12/G061 of GA ČR).

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 350–364

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Throughout the sciences, clustering plays a crucial role in data exploration and analysis. In the typical setting, we are presented with a set of data points that we wish to partition into some number of groups, called clusters, in such a way that some notion of within-cluster similarity is large and, optionally, between-cluster similarity is small. The data in question can then be well-represented by selecting one point from each cluster, typically called cluster centers. Commonly-used objectives for measuring the goodness of a clustering (and a selection of cluster centers) include k-means, k-centers and k-medians [3]. Most such objectives give rise to clustering problems that are known to be **NP**-hard (see, for example, [30] and citations therein). These problems have a rich history in computer science and engineering. Clustering problems date back at least as far as 1857 [35], and a number of clustering algorithms are well-known outside the theory community, most notably Lloyd's algorithm [32].

In the last 15 years, as data sets have come to outgrow the available memory on most machines, the streaming model of computation has emerged as a popular area of algorithmic research. In this model, computation must be performed using memory of size sublinear in the input, and (typically) using at a single pass over the data [2, 33]. Several clustering problems have been addressed in the streaming model, including k-median [25, 13, 27, 24], k-means [14, 23] and facility location [19].

While the streaming model is a sensible one for many applications, it is less suitable for some applications in domains such as network monitoring and social media [15, 17, 16, 34], where observations that have arrived more recently are in some sense more important to the computation being performed than are older observations. For example, when tracking topics in social media, a researcher may wish to have topics decay over time. The sliding window model, a variation on the streaming model, was developed to better capture these situations [20, 8, 11, 7, 10]. In this model, data arrives in a stream, and the goal is to maintain a computation only on the most recent elements. The sliding window model has received renewed attention in recent years [9, 18, 5], but no theoretical results for clustering in the sliding window model have been published since 2003 [4]. This most recent result gives a solution to the k-median clustering problem, which has been comparatively well-studied by streaming algorithm researchers. Recent years have seen impressive results yielding polylogarithmic space solutions in both the insertion-only streaming model [13, 27] and the insertion-deletion model [28, 24, 29], but to date the question of whether or not analogous results hold in the sliding window model remained open. In particular, the following question by Babcock et al. [4] has remained open for more than a decade:

"Whether it is possible to maintain approximately optimal medians in polylogarithmic space (as Charikar et al. [13] do in the stream model without sliding windows), rather than polynomial space, is an open problem."

1.1 Our Contribution

In the current work, we partially answer the question posed by Babcock, et al. [4] in the affirmative. Specifically, we give the first polylogarithmic space (α, β)-approximation algorithm for k-median clustering in the sliding window model under the assumption that the optimal k-median cost is is at most polynomial in the window size. We note that this boundedness assumption is also made by Babcock, et al. (see Lemma 5 of [4]). We justify this assumption by showing that when the optimal k-median cost is exponential in the window size, no sublinear space approximation is possible. This is in contrast to the insert-only model, where no such boundedness assumption is necessary [13, 27].

1.2 Related Work

k-median clustering of points in arbitrary metric spaces is relatively well-studied in the insertion-only streaming model. Guha, Mishra, Motwani and O'Callaghan [25] were the first to propose an insertion-only streaming algorithm for the k-median problem. They gave a $2^{O(1/\epsilon)}$-approximation streaming algorithm that uses $O(n^\epsilon)$ space, where $\epsilon < 1$. Later, Charikar, O'Callaghan, and Panigrahy [13] exponentially improved this algorithm by developing a constant factor approximation (insertion-only) streaming algorithm using $O(k \cdot \log^2 n)$ space. Their approach, somewhat similarly to [25], operates in phases, maintaining a set of $O(k \log n)$ candidate centers with the invariant that at any time during the algorithm, the candidate centers yield a suitably low clustering cost. Once the entire stream has been observed, an offline k-median clustering of this (weighted) collection of candidate centers is used as the solution for the whole stream.

The k-median clustering in the geometric setting where points are taken from a d-dimensional Euclidean space $\mathbb{R}^d$ is also well-studied in the insertion-only streaming model. In particular, Har-Peled and Mazumdar [27] used (strong) coresets to obtain a $(1+\epsilon)$-approximation algorithm for k-median and k-means problems in the insertion-only streaming model. Roughly speaking, a strong (k, ϵ)-coreset for k-median is a weighted subset S of P, so that for any set of k centers in $\mathbb{R}^d$, the weighted sum of distances from points in S to the nearest centers is approximately the same as (differs by a factor of $(1 \pm \epsilon)$ from) the sum of distances from points in P to the nearest centers. Their coreset was of size $O(k\epsilon^{-d} \log n)$. In the streaming model of computation they implemented their coreset (using the famous Merge-and-Reduce approach [6, 1]) using $O(k\epsilon^{-d} \log^{2d+2} n)$ space. Later, Har-Peled and Kushal [26] showed that one can construct (k, ϵ)-coresets for k-median with size independent of n, namely of size $O(k^2\epsilon^{-d})$. However, in the streaming model, the implementation of the new coreset (once again, using the Merge-and-Reduce approach) does not give a significant improvement in space usage. Very recently, Feldman, Fiat, and Sharir [22] extended this type of coreset for linear centers or facilities where facilities can be lines or flats.

For high-dimensional spaces, Chen [14] proposed a (k, ϵ)-coreset of size $O(k^2 d\epsilon^{-2} \log^2 n)$. in the insertion-only streaming model using $O(k^2 d\epsilon^{-2} \log^8 n)$ space. Chen's coreset works for general metric spaces as well, where he developed a technique that produces a coreset of size $O(k\epsilon^{-2} \log n(k \log n + \log(1/\delta)))$ with probability of success $1 - \delta$. If we plug the very recent 2.661-approximation algorithm for the k-median problem due to Byrka, Pensyl, Rybicki, Srinivasan, and Trinh [12] into Chen's (k, ϵ)-coreset construction, we obtain an $O(k^2\epsilon^{-2} \log(1/\delta) \log^9 n)$-space 5.322-approximation algorithm in the insertion-only streaming model with probability of success $1 - \delta$ for $0 < \delta < 1$.

To the best of our knowledge, there is no insertion-deletion streaming algorithm for k-median or k-means clustering when points are from an arbitrary metric space. However, for geometric k-median and k-means clustering, Frahling and Sohler [24] showed that they can maintain a (k, ϵ)-coreset of size $O(k\epsilon^{-d-2} \log n)$ using a different coreset construction (than [27, 26]) for data streams with insertions and deletions. This model of data streams with insertions and deletions for geometric problems was introduced by Indyk [28] and is known as *dynamic geometric data streams*. Frahling and Sohler's algorithm uses $O(k^2\epsilon^{-2d-4} \log^7 n)$ space for the k-median problem. They further showed that similar coresets exists for the geometric versions of Max-Cut, maximum weighted matching, maximum travelling salesperson, maximum spanning tree, and average distance problems, which in turn give $O(\epsilon^{-2d-4} \log^7 n)$-space streaming algorithms for these problems in data streams with insertions and deletions.

In contrast to the insertion-only streaming model and dynamic geometric streaming model, little work has been done on the k-median problem in the sliding window model,

Table 1 Known results for k-median problem in data streams. Note that the current work as well as the first algorithm of [4] give bicriteria solutions to the k-median problem, while the other results in this table return precisely k centers.

Reference	Metric space	Stream model	Approx.	Centers	Space
[25]	General	Insertion-only	$2^{O(1/\tau)}$	k	$O(n^\tau)$
[13]	General	Insertion-only	$O(1)$	k	$O(k \log^2 n)$
[14]+[12]	General	Insertion-only	5.322	k	$O(k^2\epsilon^{-2} \log(1/\delta) \log^9 n)$
[27]	Euclidean	Insertion-only	$(1+\epsilon)$	k	$O(k\epsilon^{-d} \log^{2d+2} n)$
[14]	Euclidean	Insertion-only	$(1+\epsilon)$	k	$O(k^2 d\epsilon^{-2} \log^8 n)$
[24]	Euclidean	Insertion-deletion	$(1+\epsilon)$	k	$O(k^2\epsilon^{-2d-4} \log^7 n)$
[4]	General	Sliding Windows	$2^{O(1/\tau)}$	$2k$	$O(k\tau^{-4}W^{2\tau} \log^2 W)$
[4]	General	Sliding Windows	$2^{O(1/\tau)}$	k	$O(k\tau^{-4}W^{2\tau} \log^2 W)$
This work	General	Sliding Windows	35	$2k$	$O(k^2\epsilon^{-3} \log(1/\delta) \log^{10} W)$

where we wish to produce a solution only on the most recent W elements in the data stream. The result given in [4] is, to our knowledge, the only previously existing solution in this model. The algorithm given in [4] finds an $O(2^{O(\frac{1}{\tau})})$-approximation to the k-median problem in the sliding window model for $0 < \tau < \frac{1}{2}$ using $2k$-centers and requires memory of size $O(\frac{k}{\tau^4} W^{2\tau} \log^2 W)$. With an additional step, they reduce the number of centers from $2k$ to k using the same space and with the same approximation ratio. We leave as an open problem whether a similar approach can be applied to our algorithm, which produces a bicriteria solution with between k and $2k$ centers.

Table 1 summarizes the known results for clustering problems in various streaming models.

Outline

Section 2 establishes notation and background for the remainder of the paper. Section 3 presents our main result. Section 4 proves a lower bound on the space required to find an approximate k-median solution when the optimal cost is exponential in the window size.

2 Preliminaries

We will begin by introducing some notation and basic definitions. We first define the metric k-median clustering problems. Later we will define the sliding window model, smooth functions, and smooth histograms. Finally, we will illustrate with an example that the k-median clustering is not smooth (and in fact, neither are many other clustering problems), but fortunately, we show we can compute k-median approximately in the sliding windows model, if we relax the constraint of returning exactly k centers.

2.1 Metric and Geometric k-Median Problems

Let (X, dist) be a metric space where X is a set of points and $\text{dist} : X \times X \to \mathbb{R}$ is a distance function defined over the points of X. Let $\text{dist}(p, Q) = \min_{q \in Q} \text{dist}(p, q)$ denote the distance between a point $p \in X$ and a set $Q \subseteq X$.

Let $P \subseteq X$ be a subset of points. We define $\rho_P = \min_{p,q \in P, p \neq q} \text{dist}(p, q)$ as the minimum distance between two distinct points in a set P.

▶ **Definition 2.1** (Metric k-median)**.** Let $P \subseteq X$ be a set of n points in a metric space (X, d) and let $k \in \mathbb{N}$ be a natural number. Suppose $C = \{c_1, \ldots, c_k\}$ is a set of k centers. The clustering of point set P using C is the partitioning of P such that a point $p \in P$ is in cluster C_i if $c_i \in C$ is the nearest center to p in C, that is point p is assigned to its nearest center $c_i \in C$. The cost of k-median clustering by C is $\mathsf{COST}(P, C) = \sum_{p \in P} \mathrm{dist}(p, C)$. The metric k-median problem is to find a set $C \subset P$ of k centers that minimizes the cost $\mathsf{COST}(P, C)$, that is

$$\mathsf{COST}(P, C) = \sum_{p \in P} \mathrm{dist}(p, C) = \min_{C' \subset P : |C'| = k} \mathsf{COST}(P, C') = \min_{C' \subset P : |C'| = k} \sum_{p \in P} \mathrm{dist}(p, C'),$$

where $\mathrm{dist}(p, C) = \min_{c \in C} \mathrm{dist}(p, c)$ and $\mathrm{dist}(p, C') = \min_{c \in C'} \mathrm{dist}(p, c)$

We define $\mathsf{OPT}(P, k) = \min_{C' \subset P : |C'| = k} \mathsf{COST}(P, C')$ to be the minimum k-median cost of P. Since the metric k-median problem is known to be **NP**-hard [30], we will focus on *approximation* algorithms.

▶ **Definition 2.2** ((α, β)-approximation algorithm)**.** We say an algorithm $\mathcal{A}$ is an (α, β)-approximation algorithm for the k-median problem on point set $P \subset X$ if $\mathcal{A}(P, k)$ returns a set $C \subset P$ of at most $\beta \cdot k$ centers whose cost is α-approximation of $\mathsf{OPT}(P, k)$, that is, $\mathsf{COST}(P, C) \leq \alpha \cdot \mathsf{OPT}(P, k)$.

2.2 Sliding Windows Model

Let (X, d) be a metric space. Let $P = \{p_1, p_2, \cdots, p_n\} \subseteq X$ be a point set of size $|P| = n$. In the insertion-only streaming model [2, 27, 13], we think of a (mostly adversarial) permutation $\{p'_1, p'_2, \cdots, p'_n\}$ of point set P given in a streaming fashion and the goal is to compute a function f exactly or approximately at the end of the stream using sublinear space in n, i.e, $o(n)$. Here we say point p'_t is revealed at time t.

The *sliding windows model* [20] is a generalization of the insertion-only streaming model in which we seek to compute a function f over only the W most recent elements of the stream. Given a current time t of the stream, we consider a window $\mathcal{W}$ of size W consisting of points that are inserted in the interval $[\max(t - W, 1), \cdots, t]$. Here we still assume W is large enough that we cannot store all of window $\mathcal{W}$ in memory, for example $W = \Omega(n)$; otherwise computing function f over window $\mathcal{W}$ will be trivial. A point p in the current window $\mathcal{W}$ is called *active*, and *expired*, otherwise.

2.3 Smooth Function and Smooth Histogram

Braverman and Ostrovsky [9] introduced *smooth histograms* as an effective method to compute *smooth functions* in the sliding windows model. A smooth function is defined as follows.

▶ **Definition 2.3** ((ϵ, ϵ')-smooth function [9])**.** Let f be a function, $0 < \epsilon, \epsilon' < 1$, and c be a constant number. We say f is a (ϵ, ϵ')-*smooth function* if function f is non-negative (i.e., $f(A) \geq 0$), non-decreasing (i.e., for $A \subseteq B$, $f(A) \leq f(B)$), and polynomially bounded $f(A) \leq O(|A|^c)$ such that

$$f(B) \geq (1 - \epsilon) \cdot f(A \cup B) \quad \text{implies} \quad f(B \cup C) \geq (1 - \epsilon') \cdot f(A \cup B \cup C) \ .$$

Interestingly, many functions are smooth. For instance, sum, count, min, diameter, L_p-norms, frequency moments and the length of the longest subsequence are all smooth functions.

We define $[a] = \{1, 2, 3, \cdots a\}$ and $[a, b] = \{a, a+1, a+2, \cdots b\}$ for $a \leq b$ and $a, b \in \mathbb{N}$. When there is no danger of confusion, we denote the set of points $\{p_a, p_{a+1}, \ldots, p_b\}$ as simply $[a, b]$. For example, we denote $f(\{p_a, p_{a+1}, \ldots, p_b\})$ by $f([a, b])$.

To maintain a smooth function f on sliding windows, Braverman and Ostrovsky [9] proposed a data structure that they called *smooth histograms* which is defined as follows.

▶ **Definition 2.4** (Smooth histogram [9]). Let $0 < \epsilon, \epsilon' < 1$ and $\alpha > 0$. Let f be an (ϵ, ϵ')-smooth function. Suppose there exists an *insertion-only* streaming algorithm $\mathcal{A}$ that calculates an α-approximation f' of f. The *smooth histogram* is a data structure consisting of an increasing set of indices $X_N = [x_1, x_2, \cdots, x_t = N]$ and t instances $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_t$ of $\mathcal{A}$ such that

1. p_{x_1} is expired and p_{x_2} is active or $x_1 = 0$.
2. For $1 < i < t-1$ one of the following holds
   a. $x_{i+1} = x_i + 1$ and $f'([x_{i+1}, N]) \leq (1-\epsilon')f'([x_i, N])$,
   b. $f'([x_{i+1}, N]) \geq (1-\epsilon)f'([x_i, N])$ and if $i \in [t-2]$, $f'([x_{i+2}, N]) \leq (1-\epsilon')f'([x_i, N])$.
3. $\mathcal{A}_i = \mathcal{A}([x_i, N])$ maintains $f'([x_i, N])$.

Observe that the first two elements of sequence X_N always sandwiches the current window $\mathcal{W}$ of size W, that is, $x_1 \leq N - W \leq x_2$. Braverman and Ostrovsky [9] used this observation to show that either $f'([x_1, N])$ or $f'([x_2, N])$ is a reasonably good approximation of $f'([N-W, N])$. In particular, using smooth histograms, they proved the following theorem.

▶ **Theorem 2.5.** *[9] Let $0 < \epsilon, \epsilon' < 1$ and $\alpha, \beta > 0$. Let f be an (ϵ, ϵ')-smooth function. Suppose there exists an* insertion-only *streaming algorithm $\mathcal{A}$ that calculates an α-approximation f' of f using $g(\alpha)$ space and $h(\alpha)$ update time. Then, there exists a sliding window algorithm $\mathcal{B}$ that maintains $(1 \pm (\alpha + \epsilon))$-approximation f'' of f using $O(\beta^{-1} \cdot \log n \cdot (g(\alpha) + \log n))$ space and $O(\beta^{-1} \cdot \log n \cdot h(\alpha))$ update time.*

2.4 k-median is not a smooth function

Unfortunately, it is simple to see that many clustering functions are not smooth. Here we give a simple example showing that the k-median cost is not a smooth function.

▶ **Lemma 2.6.** *k-median clustering is not a smooth function.*

Proof. We give a counterexample showing that k-median clustering is not a smooth function. Assume that we have points $p, q, r \in X$ and $k = 2$. Then $\mathsf{OPT}(\{p, q\}, k) = 0$ and $\mathsf{OPT}(\{q\}, k) = 0$, but $\mathsf{OPT}(\{q, r\}, k) = 0$ and $\mathsf{OPT}(\{p, q, r\}, k) = \min(\text{dist}(p, q), \text{dist}(p, r), \text{dist}(q, r))$ which can be arbitrarily large. ◀

However, the following lemma shows that we can compute k-median approximately in the sliding windows model if we relax the constraint of returning exactly k centers.

▶ **Lemma 2.7.** *Let $A, B, C \subset X$ be three point sets. Let $\lambda > 1$ be a parameter. Then,*

$$\mathit{OPT}(B, k) \geq \frac{1}{\lambda} \cdot \mathit{OPT}(A \cup B, k)$$

implies

$$\mathit{OPT}(B \cup C, k) \geq \frac{1}{(\lambda + 1)} \cdot \mathit{OPT}(A \cup B \cup C, 2k) \ .$$

Proof. We bound the optimal $2k$-median cost of the set $A \cup B \cup C$ as follows. First of all, the optimal $2k$-median cost of $A \cup B \cup C$ is upper-bounded by the optimal k-median cost of A plus the optimal k-median cost of $B \cup C$

$$\mathsf{OPT}(A \cup B \cup C, 2k) \leq \mathsf{OPT}(A, k) + \mathsf{OPT}(B \cup C, k) \ ,$$

as otherwise we can always replace the $2k$ optimal centers of $A \cup B \cup C$ by the k optimal centers of A and the k optimal centers of $B \cup C$ contradicting the minimum cost of the $2k$ optimal centers $A \cup B \cup C$. Now we have,

$$\begin{aligned}\mathsf{OPT}(A \cup B \cup C, 2k) &\leq \mathsf{OPT}(A, k) + \mathsf{OPT}(B \cup C, k) \leq \mathsf{OPT}(A \cup B, k) + \mathsf{OPT}(B \cup C, k) \\ &\leq \lambda \cdot \mathsf{OPT}(B, k) + \mathsf{OPT}(B \cup C, k) \\ &\leq \lambda \cdot \mathsf{OPT}(B \cup C, k) + \mathsf{OPT}(B \cup C, k) = (\lambda + 1)\mathsf{OPT}(B \cup C, k),\end{aligned}$$

which completes the proof. ◀

3 (α, β)-approximation for k-median problem on sliding windows

Here we state our main result.

▶ **Theorem 3.1.** *Let $\alpha > 1$ be a constant, $\lambda = (1 + \epsilon)$ and $k \in \mathbb{N}$ be a parameter. Let W be the size of the sliding window. Suppose the optimal k-median cost of a point set $P \subset X$ is polynomially bounded, that is $\mathsf{OPT}(P, k) = |P|^{O(c)} \cdot \rho_P$ for sufficiently large constant c, where ρ_P is the minimum distance between two distinct points in P. Suppose there exists an insertion-only streaming algorithm $\mathcal{A}(P, k)$ that maintains an α-approximation set of k centers for P using $g(\alpha)$ space and $h(\alpha)$ update time. Then there is an $[\alpha(\alpha(1 + \epsilon) + 1), 2]$-approximation sliding windows algorithm that uses $O(g(\alpha) \cdot \epsilon^{-1} \cdot \log(W \cdot \rho_P))$ space and has $O(h(\alpha) \cdot \epsilon^{-1} \cdot \log(W \cdot \rho_P))$ update time.*

Overview of Algorithm 1 (k-MEDIAN)

Suppose our stream is $S = [p_1, p_2, p_3, \cdots, p_N, \cdots, p_n]$ and we are interested to maintain a k-median clustering of a window $\mathcal{W}$ of W most recent points in stream S. We maintain an ordered list $x_i \in X = [x_1, x_2, \cdots, x_t]$ of $t = O(\epsilon^{-1} \cdot \log(n \cdot \rho_P))$ indices for $x_i \in \{1, \cdots, N\}$. Denote by $\mathcal{A}([i, j], k)$ an instance of algorithm $\mathcal{A}$ run on points $\{p_i, p_{i+1}, \cdots, p_j\}$. For every index x_i we run two instances of insertion-only streaming algorithm $\mathcal{A}(P, k)$. One instance is $\mathcal{A}([x_i, N], k)$ that maintains a set $C_{i,k}$ of k centers for the point set $\{p_{x_i}, p_{x_{i+1}, \cdots, p_N}\}$. The other instance is $\mathcal{A}([x_i, N], 2k)$, which maintains a set $C_{i,2k}$ of $2k$ centers for the point set $\{p_{x_i}, p_{x_{i+1}, \cdots, p_N}\}$.

Upon arrival of a new point p_N, we feed p_N to instances $\mathcal{A}([x_i, N], k)$ and $\mathcal{A}([x_i, N], 2k)$ for every $x_i \in X$. We also instantiate two instances $\mathcal{A}([N, N], k)$ and $\mathcal{A}([N, N], 2k)$. We then go through indices $1 \leq i \leq t - 2$ and find the greatest $j > i$ for which $\mathsf{COST}([x_j, N], C_{j,k}) \geq \frac{1}{\lambda} \cdot \mathsf{COST}([x_i, N], C_{i,k})$, and eliminate all indices x_r and their instances $\mathcal{A}$ for $i < r < j$. We then update the indices in sequence X accordingly. Finally, we find the smallest index i whose p_{x_i} is expired and $p_{x_{i+1}}$ is active. For all $r < i$, we delete x_r and its instances and update the indices in sequence X. At the end, we return the center set $C_{i,2k}$ of $2k$ centers maintained by $\mathcal{A}([x_1, N], 2k)$ as our solution.

Algorithm 1 k-MEDIAN in Sliding Windows.

Input: A stream $S = [p_1, p_2, p_3, \cdots, p_N, \cdots, p_n]$ of points from a metric space (X, dist) and the sliding window size W.

Output: A center set $C_{1,2k}$ of $2k$ centers that is an $[\alpha(\alpha(1+\epsilon)+1), 2]$-approximation of $\mathsf{OPT}(P, k)$.

Update Process, upon the arrival of new point p_N:

1: **for** $x_i \in X = [x_1, x_2, \cdots, x_t]$ ▶ (where $x_i \in \{1, \cdots, N\}$) **do**
2: Let $C_{i,k} = \mathcal{A}([x_i, N], k)$ be the set of k centers maintained by $\mathcal{A}([x_i, N], k)$, where $[x_i, N] = \{p_{x_i}, p_{x_{i+1}}, \cdots, p_N\}$.
3: Let $C_{i,2k} = \mathcal{A}([x_i, N], 2k)$ be the set of $2k$ centers maintained by $\mathcal{A}([x_i, N], 2k)$.
4: Let $t = t + 1$, $x_t = N$.
5: Let $C_{t,k} = \mathcal{A}([N, N], k)$ and $C_{t,2k} = \mathcal{A}([N, N], 2k)$, where $[N, N]$ contains only point p_N.

6: **for** $i = 1$ to $t - 2$ **do**
7: Find the greatest $j > i$ such that $\mathsf{COST}([x_j, N], C_{j,k}) \geq \frac{1}{\lambda} \cdot \mathsf{COST}([x_i, N], C_{i,k})$.
8: For $i < r < j$, delete x_r and center sets $C_{r,k}$ and $C_{r,2k}$.
9: Update the indices in sequence X accordingly.
10: Let i be the smallest index such that p_{x_i} is expired and $p_{x_{i+1}}$ is active.
11: **for** $r < i$ **do**
12: Delete x_r and center sets $C_{r,k}$ and $C_{r,2k}$.
13: Update the indices in sequence X.

Output Process:

1: Return $C_{1,2k}$ maintained by $\mathcal{A}([x_1, N], 2k)$.

Analysis

Next we prove Theorem 3.1. First we prove the approximation factor that we claim in this theorem.

▶ **Lemma 3.2.** *For assumptions of Theorem 3.1, Algorithm 1 maintains an $[\alpha(\alpha\lambda + 1), 2]$-approximation set of k centers for P in the sliding windows model, i.e.,*

$$\mathit{COST}([x_1, N'], C_{1,2k}) \leq \alpha(\alpha \cdot \lambda + 1) \cdot \mathit{OPT}(\mathcal{W}, k) \ ,$$

where $\mathcal{W}$ is the current window of size W, that is the points in the interval $[\max(N' - W, 1), N']$.

Proof. Let us fix the arrival of a new point p_N from stream $S = [p_1, p_2, p_3, \cdots, p_N, \cdots, p_n]$. From Lines (10) to (13) of Algorithm 1 we always have $x_1 \leq N - W \leq x_2$ where W is the window size. Moreover, based on Lines (6) to (9), we have

$$\mathsf{COST}([x_2, N], C_{2,k}) \geq \frac{1}{\lambda} \cdot \mathsf{COST}([x_1, N], C_{1,k}) \ .$$

Since Algorithm $\mathcal{A}(P, k)$ is an insertion-only streaming algorithm that maintains a set of k centers with an α-approximation guarantee of $\mathsf{OPT}(P, k)$, we have

$$\mathsf{OPT}([x_2, N], k) \leq \mathsf{COST}([x_2, N], C_{2,k}) \leq \alpha \cdot \mathsf{OPT}([x_2, N], k)$$

and

$$\mathsf{OPT}([x_1, N], k) \leq \mathsf{COST}([x_1, N], C_{1,k}) \leq \alpha \cdot \mathsf{OPT}([x_1, N], k) \ .$$

Therefore,

$$\alpha\mathsf{OPT}([x_2,N],k) \geq \mathsf{COST}([x_2,N],C_{2,k}) \geq \frac{1}{\lambda}\cdot\mathsf{COST}([x_1,N],C_{1,k}) \geq \frac{1}{\lambda}\cdot\mathsf{OPT}([x_1,N],k),$$

which means

$$\mathsf{OPT}([x_2,N],k) \geq \frac{1}{\alpha\cdot\lambda}\cdot\mathsf{OPT}([x_1,N],k).$$

Imagine the arrival of a new point $p_{N'}$ from the stream

$$S = [p_1,p_2,p_3,\cdots,p_N,\cdots,p_{N'},\cdots,p_n]$$

for which $x_1 \leq N'-W \leq x_2$ where $N' > N$. Denote our window by $\mathcal{W} = \{p_{N'-W},\cdots,p_{N'}\}$ and let $A = [x_1,x_2) = \{p_{x_1},p_{x_1+1},\cdots,p_{x_2-1}\}$, $B = [x_2,N] = \{p_{x_2},p_{x_2+1},\cdots,p_N\}$, and $C = \{p_{N+1},p_{N+2},\cdots,p_{N'}\}$. Observe that $A\cup B = [x_1,N] = \{p_{x_1},p_{x_1+1},\cdots,p_N\}$ and

$$B\cup C \subseteq \mathcal{W} = \{p_{N'-W},\cdots,p_{N'}\} \subseteq A\cup B\cup C = \{p_{x_1},\cdots,p_{N'}\}.$$

Now we use Lemma 2.7 which says if $\mathsf{OPT}(B,k) \geq \frac{1}{\lambda}\cdot\mathsf{OPT}(A\cup B,k)$, we then have $\mathsf{OPT}(B\cup C,k) \geq \frac{1}{(\lambda+1)}\cdot\mathsf{OPT}(A\cup B\cup C,2k)$. We replace λ by $\alpha\lambda$ to obtain

$$\begin{aligned}\mathsf{OPT}(B,k) = \mathsf{OPT}([x_2,N],k) &\geq \frac{1}{\alpha\cdot\lambda}\cdot\mathsf{OPT}(A\cup B,k)\\ &= \frac{1}{\alpha\lambda}\mathsf{OPT}([x_1,N],k) = \frac{1}{\alpha\lambda}\mathsf{OPT}(\{p_{x_1},p_{x_1+1},\ldots,p_N\},k).\end{aligned}$$

Therefore, we have

$$\begin{aligned}\mathsf{OPT}(\mathcal{W},k) &\geq \mathsf{OPT}(B\cup C,k) = \mathsf{OPT}([x_2,N'],k) = \mathsf{OPT}(\{p_{x_2},p_{x_2+1},\cdots,p_{N'}\},k)\\ &\geq \frac{1}{(\alpha\cdot\lambda+1)}\cdot\mathsf{OPT}(A\cup B\cup C,2k) = \frac{1}{(\alpha\cdot\lambda+1)}\cdot\mathsf{OPT}([x_1,N'],2k)\\ &= \frac{1}{\alpha(\alpha\cdot\lambda+1)}\cdot\mathsf{COST}([x_1,N'],C_{1,2k})\ ,\end{aligned}$$

since $\mathcal{A}([x_1,N'],2k)$ returns an α-approximation $2k$-median center set $C_{1,2k}$ of point set $[x_1,N']$, i.e.,

$$\mathsf{COST}([x_1,N'],C_{1,2k}) \leq \alpha\cdot\mathsf{OPT}([x_1,N'],2k)\ .$$

Therefore, we have

$$\mathsf{COST}([x_1,N'],C_{1,2k}) \leq (\alpha(\alpha\cdot\lambda+1))\cdot\mathsf{OPT}(\mathcal{W},k)\ ,$$

which proves the lemma. ◀

Next, we prove the space usage of Algorithm k-MEDIAN.

▶ **Lemma 3.3.** *For assumptions of Theorem 3.1, Algorithm 1 uses $O(g(\alpha)\cdot\epsilon^{-1}\cdot\log(W\cdot\rho_P))$ space and has $O(h(\alpha)\cdot\epsilon^{-1}\cdot\log(W\cdot\rho_P))$ update time.*

Proof. It is clear that the space is $O(g(\alpha)\cdot t)$ and that time is $O(h(\alpha)\cdot t)$, so it suffices to prove that $t = O(\epsilon^{-1}\log(W\cdot\rho_P))$.

As a loop invariant for Line 1 upon arrival of a new point, we will prove the bound $t < t^*$. Since Line 4 is the only place where t is incremented, we will have that $t \leq t^*$ throughout the algorithm.

In the previous iteration, the following invariant for all $i \in [1, t-2]$ is guaranteed after execution of the loop beginning on Line 6:

$$\mathsf{COST}([x_{i+2}, N], C_{i+2,k}) < \frac{1}{\lambda} \cdot \mathsf{COST}([x_i, N], C_{i,k})$$

The previous inequality is guaranteed because if it were not true, then in Line 8, x_{i+1} would have been deleted.

Note that $\mathsf{COST}([x_{t-2}, N], C_{t-2,k}) > 0$, for otherwise we conclude that $OPT([x_{t-2}, N]) = OPT([x_t, N]) = 0$ so x_{t-1} would have been deleted. The value ρ_P is the minimum distance between points in the metric space, so $\mathsf{COST}([x_{t-2}, N], C_{t-2,k}) \geq \rho_P$. By induction, we see that:

$$\mathsf{COST}([x_{t-2}, N], C_{t-2,k}) < \left(\frac{1}{\lambda}\right)^{\frac{t-i}{2}-1} \cdot \mathsf{COST}([x_i, N], C_{i,k})$$

where $i = 2, 3$ depends on the parity of t (i.e. such that t - i is an even number). The bound $\frac{t-i}{2} - 1 \geq t/2 - 3$, and therefore:

$$\lambda^{(t/2)-3} \rho_P < \mathsf{COST}([x_i, N], C_{i,k})$$

Next, we note by polynomial-boundedness that $OPT([x_i, N]) \leq W^{O(c)} \cdot \rho_P$ because Line 10 guarantees that x_2 (and thus x_i since $i \geq 2$) is not expired and thus $|[x_i, N]| \leq W$. This is the only place where polynomial-boundedness is used. Next, by the approximation-ratio of the blackbox α-approximation, we have that $\mathsf{COST}([x_i, N], C_{i,k}) \leq \alpha \cdot OPT([x_i, N])$. Putting these together, we have:

$$\lambda^{(t/2)-3} \rho_P < \mathsf{COST}([x_i, N], C_{i,k}) \leq \alpha \cdot OPT([x_i, N]) \leq \alpha \cdot W^{O(c)} \cdot \rho_P$$

Algebraic manipulation yields that $t < 6 + 2\log_\lambda(W^{O(c)} \cdot \rho_P)$. Setting $\lambda = 1 + \epsilon$, we get that $t = O(\epsilon^{-1} \log(W \cdot \rho_P))$. ◀

Now we finish the proof of Theorem 3.1.

Proof. Proof of Theorem 3.1 We set $\lambda = (1 + \epsilon)$ and let $t = O(\epsilon^{-1} \cdot \log(W \cdot \rho_P))$. Using Lemma 3.2, Algorithm 1 (k-MEDIAN) maintains an $[\alpha(\alpha\lambda + 1), 2]$-approximation set of k centers for P in the sliding windows model. For $\lambda = (1 + \epsilon)$, the approximation factor would be $\alpha(\alpha(1 + \epsilon) + 1)$ which proves the theorem. ◀

Finally, we use the approximation algorithm of Theorem 3.1 to obtain the following result.

▶ **Corollary 3.4.** *Let $\epsilon < 1/\alpha^2$ and $0 < \delta < 1$. Assume for Algorithm $\mathcal{A}(P, k)$ in Theorem 3.1, we use the combination of algorithms [12] and [14] that guarantees 5.322-approximation to the $\mathsf{OPT}(P, k)$ with probability $1 - \delta$ and uses space $O(k^2\epsilon^{-2} \log^9(n) \cdot \log(1/\delta))$. Then, the sliding windows algorithm of Theorem 3.1 is an $[35, 2]$-approximation algorithm for the k-median problem with probability $1 - \delta$ and uses $O(k^2\epsilon^{-3} \log^{10}(n) \cdot \log(1/\delta))$ space.*

Proof. As we mentioned in Section 1.2, we can plug the very recent 2.661-approximation algorithm for the k-median problem due to Byrka, Pensyl, Rybicki, Srinivasan, and Trinh [12] into (k, ϵ)-coreset construction of Chen [14] to obtain $O(k^2\epsilon^{-2} \log^9(n) \cdot \log(1/\delta))$-space 5.322-approximation algorithm in the insertion-only streaming model with probability $1 - \delta$ for $0 < \delta < 1$. We use this algorithm as an α-approximation algorithm $\mathcal{A}$ in Theorem 3.1 and set $\epsilon < 1/\alpha^2$ to have the approximation factor of

$$\alpha(\alpha(1 + \epsilon) + 1) = 5.322(5.322(1 + \frac{1}{(5.322)^2}) + 1) \leq 35 \ . \quad ◀$$

4 Lower bound for k-Median Clustering on Sliding Windows

In this section, we show that without the assumption of polynomial-boundedness, no randomized algorithm can approximate clustering in sub-linear space.

▶ **Theorem 4.1.** *Every randomized α-approximation algorithm for 2-median on sliding windows requires $\Omega(W)$-storage.*

This result will be proved by reduction from the COUNT problem, which we now define:

▶ **Definition 4.2** (COUNT Problem)**.** Given a Boolean stream $p_1p_2p_3\ldots$ (i.e for $p_i \in \{0,1\}$) and a positive integer W, the COUNT problem on sliding windows maintains the number of elements equal to 1 in the most recent W elements.

The COUNT problem was explored by Datar, Gionis, Indyk, and Motwani [20], who developed a $(1+\epsilon)$-estimator using $O(\epsilon^{-1}\log^2 n)$ space. They also gave a matching lower bound of $\Omega(\epsilon^{-1}\log^2 n)$ memory bits for any deterministic or randomized algorithm. In Theorem 4.3, we give a simple $\Omega(W)$-space lower bound for randomized algorithms that compute COUNT exactly. The proof is included here for the sake of completeness.

▶ **Theorem 4.3.** *Any randomized algorithm that exactly computes* COUNT *with probability* $2/3$ *requires $\Omega(W)$-space.*

Proof. The INDEX problem in communication complexity [31] is the following problem. Let Alice and Bob be two players. Alice has a vector $A \in \{0,1\}^n$ with entries $\{a_i\}_{i=1}^n$. Bob has an index $I \in [n]$. Together they have to identify the value of a_I using minimum communication. It is well known [31] that the INDEX problem has a $\Omega(n)$ lower bound in the one-way communication model, when Bob cannot send messages to Alice.

We provide the following simple reduction of the INDEX problem to the COUNT problem. Suppose there is a randomized streaming algorithm X that solves the COUNT problem w.p. $2/3$ and uses w bits of memory. Alice computes X on windows of length n on stream $a_1, a_2, \ldots, a_n$ and sends the memory of X to Bob. Bob continues the computation on the stream of n zeros. Thus, the input that Alice and Bob collectively create is the stream of length $2n$ with entries $p_i = a_i$ for $i \le n$ and $p_i = 0$ for $n < i \le 2n$. Bob outputs $(c_{n+I-1} - c_{n+I})$ where $c_j = \sum_{l=j-n+1}^{j} p_l$ is the number of ones in the j-th window. Indeed

$$c_{n+I-1} - c_{n+I} = \sum_{l=I}^{n+I-1} p_l - \sum_{l=I+1}^{n+I} p_l = a_I - a_{I+n} = a_I.$$

Thus, the INDEX problem can be solved using w bits, and thus we must have $w = \Omega(n)$. ◀

The reduction from COUNT to 2-median will work by using Algorithm 2 to transform a stream of Boolean values into a stream of points in one-dimensional Euclidean space. The transformation will depend on the approximation guarantee α of the 2-median algorithm. Algorithm 2 will maintain a counter j_N. It is clear that $j_N - j_{N-W+1}$ is the exact solution of COUNT in the N^{th} window. The current counter j_N is known, but storing the entire history back to j_{N-W+1} would require W bits (since the window slides, we must keep the past W values). Instead, we use a 2-median approximation as a blackbox. The 2-median algorithm will output a set of centers, and we will prove that the location of the right-most center determines j_{N-W+1}. Thus we reduce the COUNT problem to computing 2-median approximately (to any degree of approximation, possibly dependent on W).

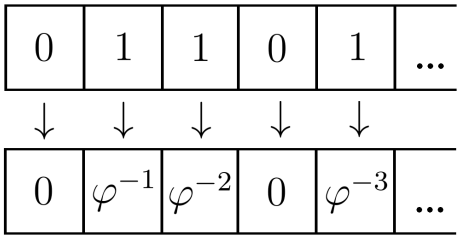


Figure 1 Example of Algorithm 2.

Algorithm 2 Streaming Transformation for an α-approximation.

$\varphi \leftarrow 4\alpha$
$j_1 \leftarrow 1$
for $N \in \{1, \ldots, n\}$ **do**
  if $p_N = 0$ **then**
    Write $x_N = 0$
    $j_{N+1} \leftarrow j_N$
  if $p_N = 1$ **then**
    Write $x_N = \phi^{-j_N}$
    $j_{N+1} \leftarrow j_N + 1$

For each $N \geq W$, define $i_N := j_{N-W+1}$. Note that the N^{th} window is completely described by i_N and j_N (henceforth denoted i and j). This set is $\{\varphi^{-i}, \ldots, \varphi^{-j+1}, 0, \ldots, 0\}$, and we assume that $k \leq j - i < W$. This assumption is valid since in the alternative cases (i.e., when $j - i \in \{0, \ldots, k-1, W\}$), both COUNT and 2-median can be solved in $O(k \log W)$-space by keeping track of the indices of non-zero terms modulo W.

Before proving the desired reduction from COUNT to 2-median in Theorem 4.5, we begin with a lemma that is essential to the argument.

▶ **Lemma 4.4.** *The optimal cost for* 2*-median is strictly less than* $2\varphi^{-i-1}$.

Proof. Consider the center set $\{0, \varphi^{-i}\}$. All points $x < \varphi^{-i}/2$ will be assigned to the center $c = 0$, and all points $x > \varphi^{-i}/2$ will be assigned to the center $c = \varphi^{-i}$. Since $\alpha \geq 1$, we have that $\varphi = 4\alpha \geq 4 > 2$ and thus all points in the window except $x = \varphi^{-i}$ have the inequality $x \leq \varphi^{-(i+1)} < \varphi^{-i}/2$. This clustering assigns the point $x = \varphi^{-i}$ to the center $c = \varphi^{-i}$ and assigns all other points to the center $c = 0$. The cost of this clustering is:

$$\sum_{a=i+1}^{j} \varphi^{-a} = \frac{\varphi^{-i} - \varphi^{-j}}{\varphi - 1} < \frac{\varphi^{-i}}{\varphi - 1}$$

$\varphi > 2$ implies $\varphi - 1 > \varphi/2$, and thus this cost is strictly less than $2\varphi^{-i-1}$. The optimum cost is bounded above by the cost of this clustering, i.e., $OPT(P, 2) < 2\varphi^{-i-1}$. ◀

▶ **Theorem 4.5.** *On sliding windows, any* α*-approximation of* 2*-median can be used to compute the exact solution of* COUNT.

Proof. We will run the α-approximation on the output of Algorithm 2 (with parameter $\varphi = 4\alpha$) and obtain a set of 2 centers for each window. We will show that the right-most center is in the interval $(\frac{1}{2}\varphi^{-i}, \frac{3}{2}\varphi^{-i})$. Because $\varphi > 3$, these intervals are disjoint for distinct i, so the right-most center will identify a unique value of i. Since we have the counter j from Algorithm 2, we may output $j - i$ as the exact solution to COUNT in that window.

All that remains to be shown is that the right-most center is in the desired interval. By Lemma 4.4, we have that $OPT(P,2) < 2\varphi^{-i-1}$. Suppose that the right-most center c_2 is not assigned any points in the clustering. Then the cost (using only the left-most center c_1) is bounded below by the optimal cost of clustering the subset $\{0, \varphi^{-i}\}$, so $COST(P,\{c_1\}) \geq OPT(\{0,\varphi^{-i}\},1) = \varphi^{-i}$. The approximation guarantee implies that:

$$\alpha \geq \frac{COST(P,\{c_1\})}{OPT(P,2)} > \frac{\varphi^{-i}}{2\varphi^{-i-1}} = \frac{\varphi}{2} = 2\alpha$$

By this contradiction, we conclude that both centers are used in the clustering, and in particular we conclude that the right-most point $x = \varphi^{-i}$ is assigned to the right-most center.

Suppose that the right-most center $c_2 \notin (\frac{1}{2}\varphi^{-i}, \frac{3}{2}\varphi^{-i})$. Then the cost of clustering the right-most point $x = \varphi^{-i}$ is at least $\frac{1}{2}\varphi^{-i}$. This implies that $COST(P,\{c_1,c_2\}) \geq COST(\{\varphi^{-i}\},\{c_2\}) \geq \frac{1}{2}\varphi^{-i}$ which leads to the following contradiction:

$$\alpha \geq \frac{COST(P,\{c_1,c_2\})}{OPT(P,2)} > \frac{\varphi^{-i}/2}{2\varphi^{-i-1}} = \frac{\varphi}{4} = \alpha$$

Therefore the rightmost-center c_2 is in the desired interval, and this completes the proof. ◀

The reduction of Theorem 4.5 together with the lower-bound of Theorem 4.3 implies the linear-space lower bound for 2-median stated in Theorem 4.1. We now give Theorem 4.6 which generalizes the reduction in two ways: (1) the result will hold for $k \geq 2$, and (2) the result will hold for clustering $f(x,c) = d(x,c)^p$ for any $p > 0$. Theorem 4.5, pertaining to 2-median, is the special case of $k = 2$ and $p = 1$. Notable special cases are k-median (when $p = 1$) and k-mean (when $p = 2$). The proof is a straight-forward generalization of the reduction for 2-median, which we briefly outline.

▶ **Theorem 4.6.** *For $k \geq 2$, every randomized α-approximation algorithm that clusters $f(x,c) = d(x,c)^p$ for $p > 0$ on sliding windows requires $\Omega(W)$-storage.*

Proof. Run Algorithm 2 with $\varphi = 2(2\alpha)^{1/p}$. A straightforward modification of Lemma 4.4 when using the center-set $\{0, \varphi^{-i-(k-2)}, \ldots, \varphi^{-i}\}$ now reads $OPT(P,k) < 2\varphi^{(-i-k+1)p}$. Supposing that only $k-1$ centers were used in the clustering, the subset $\{0, \varphi^{-i-(k-2)}, \ldots, \varphi^{-i}\}$ shows that the cost is at least $2(\varphi^{-i-(k-2)}/2)^p$. These establish a contradiction showing that the right-most point φ^i must be assigned to the right-most center. We conclude by proving that the right-most center lies in the interval $(\frac{1}{2}\varphi^{-i}, \frac{3}{2}\varphi^{-i})$ since otherwise the cost would be at least $(\varphi^{-i}/2)^p$, which again leads to the desired contradiction. ◀

Note that the transformed data was constructed in one-dimensional Euclidean space. This shows that without polynomially-boundedness, it is impossible to perform sublinear-space clustering on any Riemannian manifold, as shown in the next theorem.

▶ **Theorem 4.7.** *Let d be the metric of a Riemannian manifold M. For $k \geq 2$, every randomized α-approximation algorithm that clusters $f(x,c) = d(x,c)^p$ for $p > 0$ on sliding windows requires $\Omega(W)$-storage.*

Proof. For $\delta > 0$, let $\gamma : [0,\delta] \to M$ be a geodesic parameterized by arc-length [21]. The entire construction of Theorem 4.6 lies in the interval $[0, \varphi^{-1}] \subset [0,1]$, so we modify Algorithm 2 to output the points $\gamma(\delta\varphi^{-j})$. The proof then carries through without modification, since $d(\gamma(\delta\varphi^{-i}), \gamma(\delta\varphi^{-j})) = \delta|\varphi^{-i} - \varphi^{-j}|$ for all $i,j \geq 1$. ◀

Acknowledgements. The authors wish to thank the conference organizers for their hard work and the anonymous reviewers for their helpful feedback.

References

1. P. K. Agarwal, S. Har-Peled, and K.R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.
2. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.
3. P. Awasthi and N.F. Balcan. Center based clustering: A foundational perspective. In *Handbook of Cluster Analysis*. CRC Press, 2014.
4. B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the 9th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 234–243, 2003.
5. P. Beame, R. Clifford, and W. Machmouchi. Element distinctness, frequency moments, and sliding windows. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 290–299, 2013.
6. J.L. Bentley and J.B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.
7. V. Braverman, R. Gelles, and R. Ostrovsky. How to catch L_2-heavy-hitters on sliding windows. In *Proceedings of the 19th Computing and Combinatorics Conference*, pages 638–650, 2013.
8. V. Braverman and R. Ostrovsky. Smooth histograms for sliding windows. In *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 283–293, 2007.
9. V. Braverman and R. Ostrovsky. Effective computations on sliding windows. *SIAM Journal on Computing*, 39(6):2113–2131, 2010.
10. V. Braverman, R. Ostrovsky, and A. Roytman. Zero-one laws for sliding windows and universal sketches. In *Proceedings of APPROX-RANDOM*, pages 573–590, 2015.
11. V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.
12. J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.
13. M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 30–39, 2003.
14. K. Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.
15. G. Cormode. The continuous distributed monitoring model. *SIGMOD Record*, 42(1):5–14, 2013.
16. G. Cormode and S. Muthukrishnan. What's new: finding significant differences in network data streams. *IEEE/ACM Transactions on Networking*, 13(6):1219–1232, 2005.
17. Graham Cormode and Minos N. Garofalakis. Streaming in a connected world: querying and tracking distributed data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1178–1181, 2007.
18. M. S. Crouch, A. McGregor, and D. Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings of the 21st Annual European Symposium on Algorithms (ESA)*, pages 337–348, 2013.

19 A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler. $(1+\varepsilon)$-approximation for facility location in data streams. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1710–1728, 2013.

20 M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

21 M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, Boston, MA, 1992.

22 D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 315–324, 2006.

23 D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd Annual Symposium on Computational Geometry (SoCG)*, pages 11–18, 2007.

24 G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.

25 S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.

26 S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. In *Proceedings of the 21st Annual Symposium on Computational Geometry (SoCG)*, pages 126–134, 2005.

27 S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.

28 P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–380, 2004.

29 P. Indyk and E. Price. k-median clustering, model-based compressive sensing, and sparse recovery for earth mover distance. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 627–636, 2011.

30 K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM*, 50(6):795–824, 2003.

31 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

32 S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

33 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

34 M. Osborne, S. Moran, R. McCreadie, A. Von Lunen, M. Sykora, E. Cano, N. Ireson, C. MacDonald, I. Ounis, Y. He, T. Jackson, F. Ciravegna, and A. O'Brien. Real-time detection, tracking and monitoring of automatically discovered events in social media. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, USA, 2014.

35 J. J. Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1:79, 1857.

Congestion Games with Multisets of Resources and Applications in Synthesis

Guy Avni[1], Orna Kupferman[1], and Tami Tamir[2]

1 **School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel**
2 **School of Computer Science, The Interdisciplinary Center, Israel**

Abstract

In classical congestion games, players' strategies are subsets of resources. We introduce and study *multiset congestion games*, where players' strategies are multisets of resources. Thus, in each strategy a player may need to use each resource a different number of times, and his cost for using the resource depends on the load that he and the other players generate on the resource.

Beyond the theoretical interest in examining the effect of a repeated use of resources, our study enables better understanding of non-cooperative systems and environments whose behavior is not covered by previously studied models. Indeed, congestion games with multiset-strategies arise, for example, in production planing and network formation with tasks that are more involved than reachability. We study in detail the application of synthesis from component libraries: different users synthesize systems by gluing together components from a component library. A component may be used in several systems and may be used several times in a system. The performance of a component and hence the system's quality depends on the load on it.

Our results reveal how the richer setting of multisets congestion games affects the stability and equilibrium efficiency compared to standard congestion games. In particular, while we present very simple instances with no pure Nash equilibrium and prove tighter and simpler lower bounds for equilibrium inefficiency, we are also able to show that some of the positive results known for affine and weighted congestion games apply to the richer setting of multisets.

1998 ACM Subject Classification F.2.0 Analysis of algorithms and problem complexity – General, J.4 Social and behavioral sciences – Economics

Keywords and phrases Congestion games, Multiset strategies, Equilibrium existence and computation, Equilibrium inefficiency

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.365

1 Introduction

Congestion games model non-cooperative resource sharing among selfish players. Resources may be shared by the players and the cost of using a resource increases with the load on it. Such a cost paradigm models settings where high congestion corresponds to lower quality of service or higher delay. Formally, each resource e is associated with an increasing latency function $f_e : \mathbb{N} \to \mathbb{R}$, where $f_e(\ell)$ is the cost of a single use of e when it has load ℓ.

Previous work on congestion games assumes that players' strategies are subsets of resources, as is the case in many applications, most notably routing and network design. For example, in the setting of networks, players have reachability objectives and strategies are subsets of edges, each inducing a simple path from the source to the target [29, 3, 19]. We introduce and study multiset games, where players' strategies are multisets of resources. Thus, a player may need a resource multiple times – depending on the specific resource and strategy, and

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 365–379

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

his cost for using the resource depends on the load that he and the other players generate on it. Formally, in *multiset congestion games* (MCGs, for short), a player that uses j times a resource e that is used ℓ times by all players together, pays $j \cdot f_e(\ell)$ for these uses.

Beyond the theoretical interest in examining the effect of multisets on the extensively-studied classical games, multiset congestion games arise naturally in many applications and environments. The use of multisets enables the specification of rich settings that cannot be specified by means of subsets. We give here several examples.

As a first example, consider *network formation*. In addition to reachability tasks, which involve simple paths (and hence, subsets of resources), researchers have studied tasks whose satisfaction may involve paths that are not simple. For example, a user may want to specify that each traversal of a low-security channel is followed by a visit to a check-sum node. A well-studied class of tasks that involve paths that need not be simple are these associated with a specific length, such as patrols in a geographical region. Several communication protocols are based on the fact that a message must pass a pre-defined length before reaching its destination, either for security reasons (e.g., in *Onion routing*, where the message is encrypted in layers [27]) or for marketing purposes (e.g., advertisement spread in social networks). In addition, tasks of a pre-defined length are the components of *proof-of-work* protocols that are used to deter denial of service attacks and other service abuses such as spam (e.g., [15]), and of several protocols for sensor networks [7]. The introduction of multiset corresponds to strategies that are not necessarily simple paths [5].

In *production systems* or in *planning*, a system is modeled by a network whose nodes correspond to configurations and whose edges correspond to actions performed by resources. Users have tasks, that need to be fulfilled by taking sequences of actions. This setting corresponds to an MCG in which the strategies of the players are multisets of actions that fulfill their tasks, which indeed often involve repeated execution of actions [13]; for example "once the arm is up, do not put it down until the block is placed". Also, multiset games can model *preemptive scheduling*, where the processing of a job may split in several feasible ways among a set of machines.

Our last example, which we are going to study in detail, is *synthesis form component libraries*. A central problem in formal methods is synthesis [26], namely the automated construction of a system from its specification. In real life, hardware and software systems are rarely constructed from scratch. Rather, a system is typically constructed from a library of components by *gluing* components from a library (allowing multiple uses) [23]. For example, when designing an internet browser, a designer does not implement the TCP protocol but uses existing implementations as black boxes. The library of components is used by multiple users simultaneously, and the usages are associated with costs. The usage cost can either decrease with load (e.g., when the cost of a component represents its construction price, the users of a component share this price) as was studied in [4], or increase with load (e.g., when the components are processors and a higher load means slower performance). The later scenario induces an instance of an MCG.

Let us demonstrate the intricacy of the multiset setting with the question of the existence of a *pure Nash equilibrium* (PNE). That is, whether each instance of the game has a profile of pure strategies that constitutes a PNE – a profile such that no player can decrease his cost by unilaterally deviating from his current strategy. By [28], classical congestion games are potential games and thus always have a PNE. Moreover, by [19], in a symmetric congestion game, a PNE can be found in polynomial time. As we show in Example 1 below, a PNE might not exist in an MCG even in a symmetric two-player game over identical resources.

Table 1 Players costs. Each entry describes the cost of Player 1 followed by the cost of Player 2.

	$\{a,a,b\}$	$\{b,b,c\}$	$\{c,c,a\}$
$\{a,a,b\}$	$36,36$	$19,17$	$17,19$
$\{b,b,c\}$	$17,19$	$36,36$	$19,17$
$\{c,c,a\}$	$19,17$	$17,19$	$36,36$

Example 1: Consider the following symmetric MCG with two players and three resources: a, b, and c. The players' strategy space is $\{a,a,b\}$ or $\{b,b,c\}$ or $\{c,c,a\}$. That is, a player needs to access some resource twice and the (cyclically) consequent resource once. The latency function of all three resources is the same, specifically, $f_a(\ell) = f_b(\ell) = f_c(\ell) = \ell^2$. The players' costs in all possible profiles are given in Table 1. We show that no PNE exists in this game. Assume first that the two players select distinct strategies, w.l.o.g. $\{a,a,b\}$ and $\{b,b,c\}$. In this profile, a is accessed twice, b is accessed three times, and c is accessed once. Thus, every access of a, b and c costs $4, 9$ and 1 respectively. The cost of Player 1 is $8+9=17$, while the cost of Player 2 is $18+1=19$. By deviating to $\{c,c,a\}$, the cost of Player 2 will reduce to 17 (while the cost of Player 1 will increase to 19). Thus, no PNE in which the players select different strategies exists. If the player select the the same strategy, then one resource is accessed 4 times, and one resource is accessed twice, implying that the cost of both players is $2 \cdot 16 + 1 \cdot 4 = 36$, and any deviation is profitable. We conclude that no PNE exists in the game.

We study and answer the following questions in general and for various classes of multiset congestion games (for formal definitions, see Section 2): (i) Existence of a PNE. (ii) An analysis of *equilibrium inefficiency*. A *social optimum* (SO) of the game is a profile that minimizes the total cost of the players; thus, the one obtained when the players obey some centralized authority. It is well known that decentralized decision-making may lead to solutions that are sub-optimal from the point of view of society as a whole. We quantify the inefficiency incurred due to selfish behavior according to the *price of anarchy* (PoA) [22] and *price of stability* (PoS) [3] measures. The PoA is the worst-case inefficiency of a PNE (that is, the ratio between the cost of a worst PNE and the SO). The PoS is the best-case inefficiency of a Nash equilibrium (that is, the ratio between the cost of a best PNE and the SO). (*iii*) *Computational complexity* of finding a PNE.

Before we turn to describe our results, let us review related work. *Weighted* congestion games (WCGs, for short), introduced in [25], are congestion games in which each player i has a *weight* $w_i \in \mathbb{N}$, and his contribution to the load of the resources he uses as well as his payments are multiplied by w_i. WCGs can be viewed as a special case of MCGs, where each resource in a strategy for Player i repeats w_i times. A different extension of WCGs in which players may use a resource more than once is *integer splittable WCGs* [24, 30]. These games model the setting in which a player has a number (integer) of tasks he needs to perform and can split them among the resources. For example, in the network setting, a player might need to send $\ell \in \mathbb{N}$ packets from vertex s to t. He can send the packets on different paths, but a packet cannot be split. MCGs are clearly more general than WCGs and integer splittable WCGs – the ability to repeat each resource a different number of times lead to a much more complex setting. Thus, it is interesting to compare our results with these known for these games.

It is shown in [17, 21] that the existence of a PNE in WCGs depends on the latency function: when the latency functions are either affine or exponential, WCGs are guaranteed

to admit a PNE, whereas WCGs with a polynomial latency function need not have a PNE. In [24], the author shows that PNE always exists when the latency functions are linear using a *potential function* argument. This argument fails when the latency functions are convex, but [30] are still able to show that there is always a PNE in these games. We are able to show that the exact potential function of [17] applies also to (the much richer) affine MCGs (that is, MCGs with a affine latency function), and thus they always admit a PNE. As demonstrated in Example 1, very simple MCGs with quadratic latency functions might have no PNE.

We turn on to results in the front of equilibrium inefficiency. In congestion games with affine latency functions, both the PoA and PoS measures are well understood. It was shown in [12] that PoS $\geq 1 + \frac{1}{\sqrt{3}} \approx 1.577$ and is at most 1.6. A tight upper bound was later shown in [10]. Also, PoA $= \frac{5}{2}$ [12]. Going one step towards our setting to the study of affine WCGs, [6] shows that PoA $= 1 + \phi$, where $\phi \approx 1.618$ is the golden ratio. The PoS question is far from being settled. Only recently, [9] shows a first upper bound of 2 for PoS in linear WCGs, which is a subclass of affine WCGs. As far as we know, the only lower bound that is known for affine WCGs is the lower bound from the unweighted setting. So there is a relatively large gap between the upper- and lower-bounds for the PoS in these games.

We bound the potential function in order to show that every affine MCG G has $\text{PoS}(G) < 2$. This improves and generalizes the result in [9]. Our most technically-challenging result is the PoS lower-bound proof, which involves the construction of a family $\mathcal{G}$ of linear MCGs. Essentially, the game $G_k \in \mathcal{G}$ is parameterized by the number of players and defined recursively. The use of multisets enables us to to define a game in which, although the sharing of resources dramatically changes between its profiles, the cost a player pays is equal in all of them. For $k = 17$ we obtain that $\text{PoS}(G_{17}) > 1.631$. This is the first lower bound in these models that exceeds the 1.577 lower bound in congestion games. Finally, the PNE in $\mathcal{G}$ is achieved with dominant strategies, so our bound holds for stronger equilibrium concepts.

As for the PoA, we show that MCGs with latency functions that are polynomials of degree at most d have PoA $= \Phi_d^{d+1}$, where Φ_d is the unique nonnegative real solution to $(x+1)^d = x^{d+1}$. Observe that Φ_d is a natural generalization of the golden ratio to higher degrees. Specifically, $\Phi_1 = \phi$. For the upper bound, we adjust the upper-bound proof of [2] to our setting. We show a simplified matching lower bound; a simple two-player MCG with only two resources and latency functions of the form $f(\ell) = \ell^d$. For general latency functions we show that the PoA can be arbitrarily high.

We turn to study the application of synthesis from component libraries by multiple players. Recall that in this application, different users synthesize systems from components. A component may be used in several systems and may be used several times in a system. The quality of a system depends on the load on its components. This gives rise to an MCG, which we term a *component library game* (CLG, for short). On the one hand, a CLG is a special case of MCG, so one could expect positive results about MCGs to apply to CLGs. On the other hand, while in MCGs the strategies of the players are given explicitly by means of multisets of resources, in CLGs the strategies of the players are given symbolically by means of a specification deterministic finite automaton – the one whose language has to be composed from the library's components.

We prove that every MCG has a corresponding CLG, implying that negative results for MCGs apply to CLGs. Moreover, we show that the succinctness of the presentation of the strategies makes decision problems about MCGs more complex in the setting of CLGs. We demonstrate this by studying the complexity of the *best-response* problem – deciding whether a player can benefit from a unilateral deviation from his strategy, and the problem

of deciding whether a PNE exists in a given game. For the best-response problem, which is in P for MCGs, we prove NP-completeness for CLGs. The problem of deciding the existence of a PNE is known to be strongly NP-complete for weighted symmetric congestion games. For network congestion games with player specific cost functions, this problem is NP-complete for arbitrary networks, while a PNE can be found efficiently for constant size networks [1]. We provide a simpler hardness proof for MCGs, which is valid also for a constant number of resources, and we show that for CLGs the problem is Σ_2^P-complete. As good news, we are able to prove a "small-design property" for CLGs, which bounds the number of strategies that one needs to consider and enables us to lift to CLGs the positive results for MCGs with linear latency functions. Thus, such CLGs always have a PNE and their PoS is at most 2.

Due to the lack of space, some examples and proofs are omitted and can be found in the full version available at: `http://www.cs.huji.ac.il/~guya03/papers/FSTTCS15-full.pdf`.

2 Preliminaries

A *multiset* over a set E of elements is a generalization of a subset of E in which each element may appear more than once. For a multiset A over E and an element $e \in E$, we use $A(e)$ to denote the number of times e appears in A, and use $e \in A$ to indicate that $A(e) \geq 1$. When describing multisets, we use e^m, for $m \in \mathbb{N}$, to denote m occurrences of e.

A *multiset congestion game* (MCG) is a tuple $G = \langle K, E, \{\Sigma_i\}_{i \in K}, \{f_e\}_{e \in E} \rangle$, where $K = \{1, \ldots, k\}$ is a set of players, E is a set of *resources*, for every $1 \leq i \leq k$, the *strategy space* Σ_i of Player i is a collection of multisets over E, and for every resource $e \in E$, the *latency function* $f_e : \mathbb{N} \to \mathbb{R}$ is a non-decreasing function. The MCG G is an *affine* MCG if for every $e \in E$, the latency function f_e is affine, i.e., $f_e(x) = a_e x + b_e$, for non-negative constants a_e and b_e. Similarly, we say that G is a *linear* MCG if it is affine and for $e \in E$ we have $b_e = 0$. We assume w.l.o.g. that for $e \in E$ we have $a_e \geq 1$. Classical congestion games are a special case of MCGs where the players' strategies are sets of resources. Weighted congestion games can be viewed as a special case of MCGs, where for every $1 \leq i \leq k$, multiset $s_i \in \Sigma$ and $e \in s_i$, we have $s_i(e) = w_i$.

A profile of a game G is a tuple $P = \langle s_1, s_2, \ldots, s_k \rangle \in (\Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_k)$ of strategies selected by the players. For a resource $e \in E$, we use $L_{e,i}(P)$ to denote the number of times e is used in P by Player i. Note that $L_{e,i}(P) = s_i(e)$. We define the *load* on e in P, denoted $L_e(P)$, as the number of times it is used by all players, thus $L_e(P) = \sum_{1 \leq i \leq k} L_{e,i}(P)$ [1].

In classical congestion games, all players that use a resource e pay $f_e(\ell)$, where ℓ is the number of players that use e. As we formalize below, in MCGs, the payment of a player for using a resource e depends on the number of times he uses it. Given a profile P, a resource $e \in E$, and $1 \leq i \leq k$, the *cost of e for Player i in P* is $cost_{e,i}(P) = L_{e,i}(P) \cdot f_e(L_e(P))$. That is, for each of the $L_{e,i}(P)$ uses of e, Player i pays $f_e(L_e(P))$. The cost of Player i in the profile P is then $cost_i(P) = \sum_{e \in E} cost_{e,i}(P)$ and the cost of the profile P is $cost(P) = \sum_{1 \leq i \leq k} cost_i(P)$. We also refer to the cost of a resource e in P, namely $cost_e(P) = \sum_{i \in K} cost_{e,i}(P)$.

Consider a game G. For a profile P, player $i \in K$, and a strategy $s'_i \in \Sigma$ for Player i, let $P[i \leftarrow s'_i]$ denote the profile obtained from P by replacing the strategy for Player i by s'_i.

[1] Since our strategies are multisets, we have that $s_i(e)$, for all i and e, is an integer. Our considerations, however, are independent of this, thus all our results are valid also for games in which strategies might include fractional demands for resources. In non-splittable (atomic) games, the players must select a single strategy, even if fractional demands are allowed.

A profile P is a *pure Nash equilibrium* (PNE) if no Player i can benefit from unilaterally deviating from his strategy in P to another strategy; i.e., for every player i and every strategy $s'_i \in \Sigma$ it holds that $cost_i(P[i \leftarrow s'_i]) \geq cost_i(P)$.

We denote by OPT the cost of a social-optimal solution; i.e., $OPT = \min_P cost(P)$. It is well known that decentralized decision-making may lead to sub-optimal solutions from the point of view of society as a whole. We quantify the inefficiency incurred due to self-interested behavior according to the *price of anarchy* (PoA) [22] and *price of stability* (PoS) [3] measures. The PoA is the worst-case inefficiency of a Nash equilibrium, while the PoS measures the best-case inefficiency of a Nash equilibrium. Formally,

▶ **Definition 1.** Let $\mathcal{G}$ be a family of games, and let G be a game in $\mathcal{G}$. Let $\Upsilon(G)$ be the set of Nash equilibria of the game G. Assume that $\Upsilon(G) \neq \emptyset$.

- The *price of anarchy* of G is the ratio between the *maximal* cost of a PNE and the social optimum of G. That is, $\text{PoA}(G) = \max_{P \in \Upsilon(G)} cost(P)/OPT(G)$. The *price of anarchy* of the family of games $\mathcal{G}$ is $\text{PoA}(\mathcal{G}) = sup_{G \in \mathcal{G}} PoA(G)$.
- The *price of stability* of G is the ratio between the *minimal* cost of a PNE and the social optimum of G. That is, $\text{PoS}(G) = \min_{P \in \Upsilon(G)} cost(P)/OPT(G)$. The *price of stability* of the family of games $\mathcal{G}$ is $\text{PoS}(\mathcal{G}) = sup_{G \in \mathcal{G}} PoS(G)$.

3 Existence of a Pure Nash Equilibrium

As demonstrated in Example 1, MCGs are less stable than weighted congestion games:

▶ **Theorem 2.** *There exists a symmetric two-player MCG with identical resources and quadratic latency function that has no PNE.*

On the positive side, we show that a PNE exists in all MCGs with affine latency functions. We do so by showing that an exact potential function exists, which is a generalization of the one in [9, 18].

▶ **Theorem 3.** *Affine MCGs are potential games.*

Proof. For a profile P and a resource $e \in E$, define

$$\Phi_e(P) = a_e \cdot \Big(\sum_{i=1}^{k} \sum_{j=i}^{k} L_{e,i}(P) \cdot L_{e,j}(P)\Big) + \big(b_e \cdot \sum_{i=1}^{k} L_{e,i}(P)\big).$$

Also, $\Phi(P) = \sum_{e \in E} \Phi_e(P)$. In the full version, we prove that Φ is an exact potential function. ◀

The negative result in Theorem 2 gives rise to the decision problem ∃PNE; given an MCG, decide whether it has a PNE. Being a generalization of WCGs, the hardness results known for WCGs imply that ∃PNE is NP-hard [14]. Using the richer definition of MCGs, we show below a much simpler hardness proof. We also show hardness for games with a constant number of resources, unlike congestion games with user-specific cost functions [1].

▶ **Theorem 4.** *Given an instance of an MCG, it is strongly NP-complete to decide whether the game has a PNE, as well as to find a PNE given that one exists. For games with a constant number of resources, the problems are NP-Complete.*

▶ **Remark 5.** *In* splittable (non-atomic) games*, each player can split his task among several strategies. This can be seen as if each player is replaced by $M \to \infty$ identical players all*

having the same strategy space scaled by $1/M$. This model suits several applications, in particular planning of preemptive production. Splittable games are well-understood in classical and weighted congestion games [29, 8]. In the full version we define the corresponding MCG and show that the positive PNE-existence result, known for weighted congestion games, carry over to games with multisets of resources.

4 Equilibrium Inefficiency in MCGs

4.1 The Price of Stability

The PoS problem in affine congestion games is settled: [12, 10] show that PoS $= 1+\frac{1}{\sqrt{3}} \approx 1.577$. For affine WCGs, the problem was open for a long time, and only recently progress was made by [9], who showed that PoS ≤ 2 for linear WCGs. As far as we know, there is no known lower bound for linear WCGs that exceeds the 1.577 bound for unweighted games. We show that every affine MCG G has PoS$(G) < 2$. Thus, we both improve the result to include affine functions, tighten the bound, and generalize it. For the lower bound, we show a family of linear MCGs $\mathcal{G}$ that has PoS$(\mathcal{G}) > 1.631$. We start with the upper bound.

▶ **Theorem 6.** *Every affine MCG G has PoS$(G) < 2$.*

Proof. Consider an affine MCG G and a profile P. It is not hard to see that for the potential function Φ that is presented in Theorem 3 we have $\Phi(P) \leq cost(P)$. Moreover, for $e \in E$ we have $2\Phi_e(P) = cost_e(P) + a_e \sum_{1\leq i\leq k} L^2_{e,i}(P) + b_e \sum_{1\leq i\leq k} L_{e,i}(P)$. Thus, $\Phi(P) > \frac{1}{2}cost(P)$. The theorem follows using standard techniques: $cost(O) \geq \Phi(O) \geq \Phi(N) > \frac{1}{2}cost(N)$, where O is the social optimum and N is a PNE that is reached from O by a sequence of best-respond moves of the players. Then, $PoS(G) \leq \frac{cost(N)}{cost(O)} < 2$. The details of the proof can be found in the full version. ◀

Note that while the PoS can get arbitrarily close to 2, it is strictly smaller than 2 for every game instance. The proof in [9], on the other hand, only shows PoS ≤ 2 for the family of affine MCGs, and our result does not improve this bound.

For the lower bound, we show a family of linear MCG $\mathcal{G} = \{G_k\}_{k\geq 2}$ that are parameterized by the number of players. Using a computerized simulation, we obtain that for the game with 17 players, we have PoS$(G_{17}) > 1.631$. We leave open the problem of calculating the exact value the PoS tends to as the number of players increases. In the full version we show a graph of the PoS as a function of k, which hints that the answer is only slightly higher than 1.631.

The PNE in the games in the family is achieved with dominant strategies, and thus it is resistant to stronger types of equilibria.

▶ **Theorem 7.** *There is a linear MCG G with PoS$(G) > 1.631$.*

Proof. We define a family of games $\{G_k\}_{k\geq 2}$ as follows. The game G_k is played by k players, thus $K_k = \{1, \ldots, k\}$. For Player 1, all strategies $\Sigma^k_1 = \{O^k_1\}$ consists of a single multiset. For ease of presentation we sometimes refer to O^k_1 as N^k_1. For $i \geq 2$, the strategy space of Player i consists of two multisets, $\Sigma^k_i = \{O^k_i, N^k_i\}$. We define G_k so that for all $k \geq 2$, the profile $\bar{O}_k = \langle O^k_1, \ldots, O^k_k\rangle$ is the social optimum and the profile $\bar{N}_k = \langle N^k_1, \ldots, N^k_k\rangle$ is the only PNE.

When describing the games in the family, we partition the resources into *types* and describe a multiset as a collection of triples. A triple $\langle t, y, l\rangle$ stands for y different resources of type t, each appearing l times. For example, $\{\langle a, 2, 1\rangle, \langle b, 1, 3\rangle, \langle c, 2, 2\rangle\}$ stands for the

multiset $\{a_1, a_2, b_1, b_1, b_1, c_1, c_1, c_2, c_2\}$. In all games and resources, there are two types of latency functions; the identity function, or identity *plus* epsilon, where the second type of function are linear functions of the form $f(x) = (1+\epsilon)\cdot x$, for some $\epsilon > 0$. The latency function of resources of the same type is the same, and we use the terms "a has identity latency" and "b has identity plus ϵ latency" to indicate that all the resources a' of type a have $f_{a'}(j) = j$ and all the resources b' of type b have $f_{b'}(j) = (1+\epsilon)\cdot j$, for all numbers j of uses.

The definition of G_k is complicated and we start by describing the idea in the construction of G_2 and G_3. In the full version we also describe G_4. We start by describing G_2. The game G_2 is defined with respect to two types of resources, a and b, with identity and identity plus ϵ latency, respectively. We define Player 1's strategy space $\Sigma_1^2 = \{O_1^2\}$ and Player 2's strategy space $\Sigma_2^2 = \{O_2^2, N_2^2\}$, with $O_1^2 = N_2^2 = \langle a, 2, 1\rangle$ and $O_2^2 = \langle b, 1, 2\rangle$. That is, $\Sigma_1^2 = \{\{a_1, a_2\}\}$ and $\Sigma_2^2 = \{\{a_1, a_2\}, \{b_1, b_1\}\}$. Clearly, the profile $\bar{N_2} = \langle O_1^2, N_2^2\rangle$ is the only PNE in G_2.

We continue to describe G_3. The game G_3 is defined with respect to four types of resources, a, b, c^1 and c^2, where b has identity plus ϵ latency, c^1 has identity plus ϵ' latency, and the other resources have identity latency. Let $x_3 = 3! = 6$. We define $\Sigma_1^3 = \{O_1^3\}$, $\Sigma_2^2 = \{O_2^3, N_2^3\}$, and $\Sigma_3^3 = \{O_3^3, N_3^3\}$, with $O_1^3 = N_2^3 = \langle a, x_3, 1\rangle$, $O_2^3 = \langle b, \frac{x_3}{2}, 2\rangle$, $O_3^3 = \{\langle c^1, \frac{x_3}{3}, 3\rangle, \langle c^2, \frac{x_3}{2}, 1\rangle\}$, and $N_3^3 = \{\langle b, \frac{x_3}{2}, 1\rangle, \langle a, x_3, 1\rangle\}$. We claim that $\bar{N_3} = \langle O_1^3, N_2^3, N_3^3\rangle$ is the only PNE. Our goal here is not to show a complete proof, but to demonstrate the idea of the construction. It is not hard to see that Player 2 deviates to N_2^3 from the profile $\bar{O_3} = \langle O_1^3, O_2^3, O_3^3\rangle$, Player 3 deviates from the resulting profile $\bar{N_3} = \langle O_1^3, N_2^3, N_3^3\rangle$. The crux of the construction is to keep Player 2 from deviating back from $\bar{N_3}$. Note that since Player 3 uses the b-type resources once in $\bar{N_3}$, when Player 2 deviates from N_2^3 to O_2^3, their load increases to 3. Thus, $cost_2(\bar{N_3}[2 \leftarrow O_2^3]) = 3(3\cdot 2\cdot(1+\epsilon)) > 6(3\cdot 1) = cost_2(\bar{N_3})$ and the deviation is not beneficial.

We define the game G_k, for $k \geq 2$, as follows. Let $x_k = k!$. Player 1's strategy space consists of a single multiset $O_1^k = \langle e_{1,1}, x_k, 1\rangle$. For $2 \leq i \leq k$, assume we have defined the strategies and resources for players $1, \ldots, i-1$. We define Player i's strategies as follows. We start with the multiset N_i^k, which does not introduce new resources. We define $N_i^k = \cup_{1\leq j\leq i-1}\{\langle t, x, 1\rangle : \langle t, x, l\rangle \subseteq O_i^k\}$. The definition of O_i^k is more involved, but the idea is simple. We define O_i^k so that it satisfies two properties. First, O_i^k uses new resources. That is, for every $1 \leq j \leq i-1$, both $O_i^k \cap O_j^k = \emptyset$ and $O_i^k \cap N_j^k = \emptyset$. Consider the profile P_i in which, for every $1 \leq j < i$, Player j uses N_j^k and, for every $i \leq l \leq k$, Player l uses O_l^k. We define O_i^k so that when all resources have identity latency, $cost_i(P_i) = cost_i(P_i[i \leftarrow N_i^k])$. For every multiset $\langle e_{j,a}, x_{j,a}, 1\rangle$ in N_i^k, which we have just defined, we introduce a multiset $\langle e_{i,b}, x_{i,b}, l_{i,b}\rangle$ in O_i^k that uses new resources, where b is a unique index that is arbitrarily chosen, and x_b^i and l_b^i are defined as follows. Let $l = |\{j : e_{j,a} \in N_j^k\}|$. We define $l_{i,b} = l+1$ and $x_{i,b} = x_{j,a}/l_{i,b}$. Since O_i^k uses new resources, showing the first property is easy. In the full version we show it satisfies a much stronger property.

▶ **Claim 8.** *Consider $k \in \mathbb{N}$, a profile P in G_k, and $1 < i \leq k$. Assume Player i plays O_k^i in P. When the latency functions are identity, we have $cost_i(P) = cost_i(P[i \leftarrow N_k^i])$.*

To complete the construction, we define the latency functions so that for every $2 \leq i \leq k$, we have that $e_{i,1}$-type resources have identity plus ϵ_i latency for $0 < \epsilon_2 < \ldots < \epsilon_k$. By Claim 8 there are such values that make N_i^k a dominant strategy for Player i. Thus, the only PNE in G_k, for $k \geq 2$, is the profile $\bar{N_k} = \langle O_1^k, N_2^k, \ldots, N_k^k\rangle$. Next, we identify the social optimum.

▶ **Claim 9.** *The profile $\bar{O_k} = \langle O_1^k, \ldots, O_k^k\rangle$ is the social optimum.*

Once we identify $\bar{O_k}$ as the social optimum and $\bar{N_k}$ as the only PNE, the calculation of the PoS boils down to calculating their costs, which we do using a computer. Specifically,

we have $\text{PoS}(G_{17}) = 1.6316$, and we depict the values of G_k, for $2 \leq k \leq 17$, in the full version. ◀

▶ **Remark 10.** *We conjecture that the correct value for the PoS is closer to our lower bound of* 1.631 *rather than to the upper bound of* 2*. In the full version we show a more careful analysis of the potential function than the one in Theorem 6 that shows that for every linear MCG G we have $PoS(G) \leq 2 - \frac{\sum_{e \in E} \sqrt{cost_e(N_G)}}{cost(O_G)}$, where N_G and O_G denote the cheapest PNE and the social optimum of G, respectively. Also, we show that for every $n \geq 2$, for the MCG G_n that is described in Theorem 7, the inequality in the expression is essentially an equality.*

▶ **Remark 11.** *We can alter the family in Theorem 7 to have quadratic latency functions instead of identity functions. Although Claim 8 does not hold in the altered family, a computerized simulation shows that the N strategies are still dominant strategies. Also, using a computerized simulation, we show that the PoS for G_{15} is* 2.399*, higher than the upper bound of* 2.362 *for congestion games, which is shown in [9, 11].*

4.2 The Price of Anarchy

In this section we study the PoA for MCGs. We start with MCGs with polynomial latency functions and show that the upper bound proven in [2] for WCGs can be adjusted to our setting. Being a special case of MCGs, the matching lower bound for WCGs applies too. Still, we present a different and much simpler lower-bound example, which uses a two-player singleton MCG. In a singleton game, each strategy consists of (multiple accesses to) a single resource. Finally, when the latency functions are not restricted to be polynomials, we show that the PoA is unbounded, and it is unbounded already in a singleton MCG with only two players.

We start by showing that the PoA in polynomial MCGs is not higher than in polynomial WCGs. The proof adjusts the one known for WCGs [2] to our setting. For $d \in \mathbb{N}$, we denote by $\mathcal{P}_d$ the set of polynomials of degree at most d.

▶ **Theorem 12.** *The PoA in MCGs with latency functions in $\mathcal{P}_d$ is at most Φ_d^{d+1}, where Φ_d is the unique nonnegative real solution to $(x+1)^d = x^{d+1}$.*

Next, we show a matching lower bound that is stronger and simpler than the one in [2].

▶ **Theorem 13.** *For $d \in \mathbb{N}$, the PoA in two-player singleton MCG with latency functions in $\mathcal{P}_d$ is at least Φ_d^{d+1}.*

Proof. Let $d \in \mathbb{N}$. Consider the two-player singleton MCG G with resources $E = \{e_1, e_2\}$, strategy spaces $\Sigma_1 = \{e_1^x, e_2^y\}$ and $\Sigma_2 = \{e_1^y, e_2^x\}$, and for $\ell \in \mathbb{R}$, we define the latency functions $f_{e_1}(\ell) = f_{e_2}(\ell) = \ell^d$. We define $x = \Phi_d$ and $y = 1$. Since $x > y$ the social optimum is attained in the profile $\langle e_1^y, e_2^y \rangle$ and its cost is $2y^d = 2$. Recall that in MCGs, the players' strategies are multisets. In particular, x should be a natural number. To fix this, we consider a family of MCGs in which the ratio between x and y tends to the ratio above.

We claim that the profile $N = \langle e_1^x, e_2^x \rangle$ is a PNE. This would imply that $\text{PoA}(G) = \frac{2x^{d+1}}{2} = \Phi_d^{d+1}$, which would conclude the proof. We continue to prove the claim. The cost of a player in N is $x \cdot x^d = x^{d+1}$ and by deviating, the cost changes to $y \cdot (x+y)^d = (x+1)^d$. Our definition of x implies that $x^{d+1} = (x+1)^d$. Thus, the cost does not change after deviating. Since the players are symmetric, we conclude that the profile N is a PNE, and we are done. ◀

Finally, by taking variants with factorial latency functions to the game described in Theorem 13, we are able to increase the PoA in an unbounded manner.

▶ **Theorem 14.** *The PoA in two-player MCGs is unbounded.*

5 Synthesis from Component Libraries

In this section we describe the application of MCGs in synthesis from component libraries. As briefly explained in Section 1, in this application, different users synthesize systems by gluing together components from a component library. A component may be used in several systems and may be used several times in a system. The performance of a component and hence the system's quality depends on the load on it. We describe the setting in more detail, formalize it by means of MCGs, and relate to the results studied in earlier sections.

Today's rapid development of complex and safety-critical systems requires reliable verification methods. In *formal methods*, we reason about systems and their specifications by solving mathematical questions about their models. A central problem in formal methods is synthesis, namely the automated construction of a system from its specification. In real life, systems are rarely constructed from scratch. Rather, a system is typically constructed from a library of components by *gluing* components from the library [23]. In this setting, the input to the synthesis problem is a specification and a library of components, and the goal is to construct from the components a system that exhibits exactly the behaviors specified in the specification.

▶ **Remark 15.** *The above setting corresponds to* closed systems*, whose behavior is independent of their environment. It is possible to generalize the definitions to* open systems*, which interact with their environment. In [4], we studied both the closed and open settings in the context of cost-sharing (rather than congestion) games. The technical challenges that have to do with the system being open are orthogonal to these that arise from the congestion effects, and on which we focus in this work.*

In our setting, we use deterministic finite automata (DFAs, for short) to model the specification and use *box-DFAs* to model the components in the library. Formally, a DFA is $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is an alphabet, Q is a set of states, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $q_0 \in Q$ is an initial states, and $F \subseteq Q$ is a set of accepting states. The *run* of $\mathcal{A}$ on a word $w = w_1, \ldots w_n \in \Sigma^*$ is the sequence of states $r = r_0, r_1, \ldots, r_n$ such that $r_0 = q_0$ and for every $0 \leq i \leq n-1$, we have $r_{i+1} = \delta(r_i, w_{i+1})$. Now, a box-DFA $\mathcal{B}$ is a DFA augmented with a set of *exit states*. When a run of $\mathcal{B}$ reaches an exit state, it moves to another box-DFA, as we formalize below.

The input to the synthesis from component libraries problem is a specification DFA $\mathcal{S}$ over an alphabet Σ and a library of box-DFAs components $\mathcal{L} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$. The goal is to produce a *design*, which is a recipe to compose the components from $\mathcal{L}$ to a DFA. A design is correct if the language of the system it induces coincides with that of the specification.

Intuitively, the design can be thought of as a scheduler; it passes control between the different components in $\mathcal{L}$. When a component $\mathcal{B}_i$ is in *control*, it reads letters in Σ, visits the states of $\mathcal{B}_i$, follows its transition function, and if the run terminates, it is accepting iff it terminates in one of $\mathcal{B}_i$'s accepting states. A component relinquishes control when the run reaches one of its exit states. It is then the design's duty to choose the next component, which gains control through its initial state.

Formally (see an example in Figure 1), a *transducer* is a DFA that has, in addition to the input alphabet that labels the transitions, also an output alphabet that labels the states.

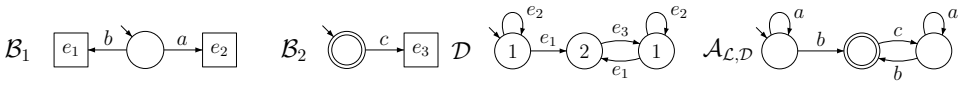


Figure 1 An example of a library $\mathcal{L} = \{\mathcal{B}_1, \mathcal{B}_2\}$, a design $\mathcal{D}$, and the resulting composition $\mathcal{A}_{\mathcal{L},\mathcal{D}}$.

Also, a transducer has no rejecting states. Let $[n] = \{1, \ldots, n\}$. A design is a transducer $\mathcal{D}$ whose input alphabet is the set $\mathcal{E}$ of all exit states of all the components in $\mathcal{L}$ and whose output alphabet is $[n]$. We can think of $\mathcal{D}$ as running beside the components. When a component reaches an exit state e, then $\mathcal{D}$ reads the input letter e, proceeds to its next state, and outputs the index of the component to gain control next. Note that the components in the library are black boxes: the design $\mathcal{D}$ does not read the alphabet Σ of the components and has no information about the states that the component visits. It only sees which exit state have been reached. Given a library $\mathcal{L}$ and a design $\mathcal{D}$, their *composition* is a DFA $\mathcal{A}_{\mathcal{L},\mathcal{D}}$ obtained by composing the components in $\mathcal{L}$ according to $\mathcal{D}$. We say that a design $\mathcal{D}$ is *correct* with respect to a specification DFA $\mathcal{S}$ iff $L(\mathcal{A}_{\mathcal{L},\mathcal{D}}) = L(\mathcal{S})$. In the full version we construct $\mathcal{A}_{\mathcal{L},\mathcal{D}}$ formally.

For example, consider the library $\mathcal{L} = \{\mathcal{B}_1, \mathcal{B}_2\}$ over the alphabet $\Sigma = \{a, b, c\}$, and the design $\mathcal{D}$ that are depicted in Figure 1. We describe the run on the word bc. The component that gains initial control is $\mathcal{B}_1$ as the initial state of $\mathcal{D}$ outputs 1. The run in $\mathcal{B}_1$ proceeds with the letter b to the exit state e_1 and relinquishes control. Intuitively, control is passed to the design that advances with the letter e_1 to the state that outputs 2. Thus, the component $\mathcal{B}_2$ gains control, and it gains it through its initial state. Then, the letter c is read, $\mathcal{B}_2$ proceeds to the exit state e_3 and relinquishes control. The design advances with the letter e_3 to a state that outputs 1, and control is assigned to $\mathcal{B}_1$. Since the initial state of $\mathcal{B}_1$ is rejecting, the word ab is rejected. As a second example, consider the word ab. Again, $\mathcal{B}_1$ gains initial control. After visiting the exit state e_2, control is reassigned to $\mathcal{B}_1$. Finally, after visiting the state e_1, control is assigned to $\mathcal{B}_2$, where the run ends. Since the initial state of $\mathcal{B}_2$ is accepting, the run is accepting.

The synthesis problem defined above is aimed at synthesizing correct designs. We now add costs to the setting. A *component library game* (CLG, for short) is a tuple $\langle K, \mathcal{L}, \{\mathcal{S}_i\}_{i \in K}, \{f_{\mathcal{B}}\}_{\mathcal{B} \in \mathcal{L}} \rangle$, where $K = \{1, \ldots, k\}$ is a set of players, $\mathcal{L}$ is a collection of box-DFAs, the objective of Player $i \in K$ is given by means of a specification DFA $\mathcal{S}_i$, and, as in MCGs, the latency function $f_{\mathcal{B}}$ for a component $\mathcal{B} \in \mathcal{L}$ maps the load on $\mathcal{B}$ to its cost with this load. For $i \in K$, the set of strategies for Player i is the set of designs that are correct with respect to $\mathcal{S}_i$. A CLG corresponds to an MCG with a slight difference that there might be infinitely many correct designs. Consider a profile $P = \langle \mathcal{D}_1, \ldots, \mathcal{D}_k \rangle$. For a component $\mathcal{B} \in \mathcal{L}$, we use $L_{\mathcal{B},i}(P)$ to denote the number of times Player i uses $\mathcal{B}$ in P. Recall that each state in the transducer $\mathcal{D}_i$ is labeled by a component in $\mathcal{L}$. We define $L_{\mathcal{B},i}(P)$ to be the number of states in $\mathcal{D}_i$ that are labeled with $\mathcal{B}$. The rest of the definitions are as in MCGs.

We first show that every MCG can be translated to a CLG:

▶ **Theorem 16.** *Consider a k-player MCG G. There is a CLG G' between k players with corresponding profiles. Formally, there is a one-to-one and onto function f from profiles of G to profiles of G' such that for every profiles P in G and Player $i \in [k]$, we have that $cost_i(P) = cost_i(f(P))$.*

Proof. Consider an MCG $\langle K, E, \{\Sigma_i\}_{i \in K}, \{f_e\}_{e \in E} \rangle$. Recall that Σ_i is the set of strategies for Player i that consists of multisets over E. We construct a CLG with alphabet $E \cup \bigcup_{i \in K} \Sigma_i$. For

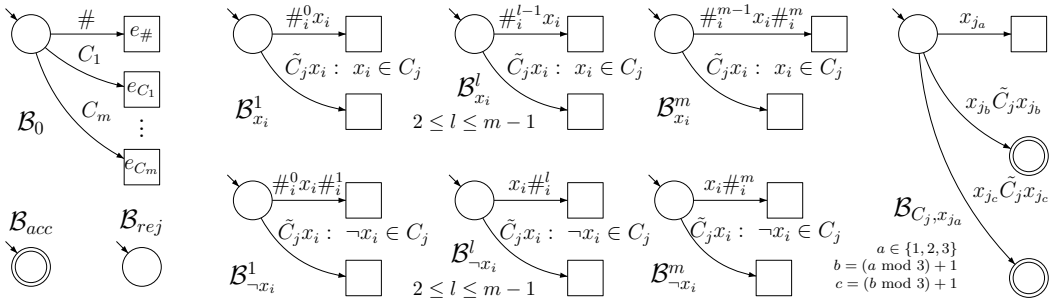


Figure 2 The components in the library $\mathcal{L}$.

$i \in K$, the specification $\mathcal{S}_i$ for Player i consists of $|\Sigma_i|$ words. Every strategy $s = \{e_1, \ldots, e_n\}$ (allowing duplicates) in Σ_i contributes to $L(S)$ the word $s \cdot e_1 \cdot e_2 \cdot \ldots \cdot e_n$. We construct a library $\mathcal{L}$ with $|E| + \sum_{i \in K} |\Sigma_i|$ components of two types: a *strategy component* $\mathcal{B}_s$ for each $s \in \Sigma_i$ and a *resource component* $\mathcal{B}_e$ for each $e \in E$. In addition, $\mathcal{L}$ contains the component $\mathcal{B}_{acc}$ that is depicted in Figure 2. Intuitively, a correct design must choose one strategy component $\mathcal{B}_s$ and then use the component $\mathcal{B}_e$ the same number of times e appears in s. We continue to describe the components. For $s \in \Sigma_i$, the component $\mathcal{B}_s$ relinquishes control only if the letter s is read. It accepts every word in $L(\mathcal{S}_i)$ that does not start with s. For $e \in E$, the resource component $\mathcal{B}_e$ has an initial state with an e-labeled transition to an exit state. Finally, the latency function for the resource components coincides with latency functions of the resources in the MCG, thus for $e \in E$, we have $f_{\mathcal{B}_e} = f_e$. The other latency functions are $f \equiv 0$. In the full version we prove that there is a cost-preserving one-to-one and onto correspondence between correct designs with respect to $\mathcal{S}_i$ and strategies in Σ_i, implying the existence of the required function between the profiles. ◀

Theorem 16 implies that the negative results we show for MCGs apply to CLGs:

▶ **Corollary 17.** *There is a CLG with quadratic latency functions with no PNE; for CLGs with affine latency functions, we have $PoS(CLG) > 1.631$; for $d \in \mathbb{N}$, the PoA in a two-player singleton MCG with latency functions in $\mathcal{P}_d$ is at least Φ_d^{d+1}.*

▶ **Remark 18.** *We note that the positive results for CLGs with linear latency functions, namely existence of PNE and $PoS(CLG) \le 2$, do not follow immediately from Theorem 3, as its proof relies on the fact that an MCG has only finitely many profiles. Since the strategy space of a player might have infinitely many strategies, a CLG might have infinitely many profiles. In order to show that CLGs with linear latency functions have a PNE we need Lemma 19 below, which implies that even in games with infinitely many profiles, there is a best response dynamics that only traverses profiles with "small" designs. Such a traversal is guaranteed to reach a PNE as there are only finitely many such profiles.* ◀

Computational complexity. We turn to study two computational problems for CLGs: finding a *best-response* and deciding the existence of a PNE. We show that the succinctness of the representation of the objectives of the players in CLGs makes these problems much harder than for MCGs. Our upper bounds rely on the following lemma. The lemma is proven in [4] for cost-sharing games, and the considerations in the proof there applies also for congestion games.

▶ **Lemma 19.** *Consider a library $\mathcal{L}$, a specification $\mathcal{S}$, and a correct design $\mathcal{D}$. There is a correct design $\mathcal{D}'$ with at most $|\mathcal{S}| \cdot |\mathcal{L}|$ states, where $|\mathcal{L}|$ is the number of states in the components of $\mathcal{L}$, such that for every component $\mathcal{B} \in \mathcal{L}$, the number of times $\mathcal{D}'$ uses $\mathcal{B}$ is at most the number of times $\mathcal{D}$ uses $\mathcal{B}$.*

We start with the best-response problem (BR problem, for short): Given an MCG $\mathcal{G}$ between k players, a profile P, an index $i \in K$, and $\mu \in \mathbb{R}$, decide whether Player i has a strategy S'_i such that $cost_i(P[i \leftarrow S'_i]) \leq \mu$.

▶ **Theorem 20.** *The BR problem for MCGs is in P. For CLGs it is NP-complete, and NP-hardness holds already for games with one player and linear latency functions.*

Proof. Showing that the BR problem is in P for MCGs follows easily from the fact the set of strategies for Player i is given implicitly and calculating the cost for a player in a profile can be done in polynomial time.

The upper bound for CLGs follows from Lemma 19, which implies an upper bound on the size of the cheapest correct designs. Since checking whether a design is correct and calculating its cost can both be done in polynomial time, membership in NP follows.

We continue to the lower bound. We describe the intuition of the reduction and the formal definition along with the correctness proof can be found in the full version. Given a 3SAT formula φ with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$, we construct a library $\mathcal{L}$ and a specification $\mathcal{S}$ such that there is a design $\mathcal{D}$ that costs at most $\mu = nm + m$ iff φ is satisfiable. The library $\mathcal{L}$ consists of an *initial component* $\mathcal{B}_0$, *variable components* $\mathcal{B}^j_{x_i}$ and $\mathcal{B}^j_{\neg x_i}$ for $j \in [m]$ and $i \in [n]$, *clause components* $\mathcal{B}_{C_j,x_{j_k}}$ for $j \in [m]$ and $k \in \{1,2,3\}$, and component $\mathcal{B}_{acc}$ and $\mathcal{B}_{rej}$. The components of the library are depicted in Figure 2. The latency function of the variable components is the identity function $f(x) = x$, thus using such a component once costs 1. The latency functions of the other components is the constant function $f \equiv 0$, thus using such components any number of times is free.

Intuitively, a correct design corresponds to an assignment to the variable and must use nm variable components as follows. For $i \in [n]$, either use all the components $\mathcal{B}^1_{x_i}, \ldots, \mathcal{B}^m_{x_i}$ or all the components $\mathcal{B}^1_{\neg x_i}, \ldots, \mathcal{B}^m_{\neg x_i}$ with a single use each. Thus, a correct design implies an assignment $\eta : \{x_1, \ldots, x_n\} \to \{T, F\}$. Choosing $\mathcal{B}^1_{x_i}, \ldots, \mathcal{B}^m_{x_i}$ corresponds to $\eta(x_i) = F$ and choosing $\mathcal{B}^1_{\neg x_i}, \ldots, \mathcal{B}^m_{\neg x_i}$ corresponds to $\eta(x_i) = T$.

Additionally, in order to verify that a correct design corresponds to a satisfying assignment, it must use m clause components and m more variable components as follows. Consider a correct design $\mathcal{D}$, and let $\eta : \{x_1, \ldots, x_n\} \to \{T, F\}$ be the corresponding assignment as described above. For every $j \in [m]$, $\mathcal{D}$ must use a clause component $\mathcal{B}_{C_j,x_i}$, where recall that the clause C_j includes a literal $\ell \in \{x_i, \neg x_i\}$. Using the component $\mathcal{B}_{C_j,x_i}$ requires $\mathcal{D}$ to use a variable component $\mathcal{B}^t_\ell$, for some $t \in [m]$. So, a correct design uses a total of $nm + m$ components with identity latency. If $\eta(\ell) = F$, then $\mathcal{B}^t_\ell$ is already in use and a second use will cost more than 1, implying that the design costs more than $nm + m$. ◀

The next problem we study is deciding the existence of a PNE. As we show in Theorem 4, the problem is NP-complete for MCGs. As we show below, the succinctness of the representation makes this problem harder for CLGs.

▶ **Theorem 21.** *The ∃PNE problem for CLGs is Σ^P_2-complete.*

Proof. The upper bound is easy and follows from Lemma 19.

For the lower bound we show a reduction from the complement of *not all equal* ∀∃ *3SAT* (NAE, for short), which is known to be Σ^P_2-complete [16]. An input to NAE is a

3CNF formula φ over variables $x_1, \ldots, x_n, y_1, \ldots, y_n$. It is in NAE if for every assignment $\eta : \{x_1, \ldots, x_n\} \to \{T, F\}$ there is an assignment $\rho : \{y_1, \ldots, y_n\} \to \{T, F\}$ such that every clause in φ has a literal that gets value truth and a literal that gets value false (in η or ρ, according to whether the variable is an x or a y variable). We say that such a pair of assignments $\langle \eta, \rho \rangle$ is *legal* for φ.

Given a 3CNF formula φ, we construct a CLG G with three players such that $\varphi \in$ NAE iff G does not have a PNE. We describe the intuition of the reduction. The details can be found in the full version. There is a one-to-one correspondence between Player 3 correct designs and assignments to the variables $\{x_1, \ldots, x_n\}$. For an assignment $\eta : \{x_1, \ldots, x_n\} \to \{T, F\}$ we refer to the corresponding correct design by $\mathcal{D}_\eta$. Consider a legal pair of assignments $\langle \eta, \rho \rangle$, and assume Player 3 chooses the design $\mathcal{D}_\eta$. Similarly to the proof of Theorem 20, the library contains variable components with identity latency function. We construct the library and the players' objectives so that there is a correct design $\mathcal{D}_\rho$ for Player 1 that uses $mn+2m$ variable components each with load 1 iff $\langle \eta, \rho \rangle$ is a legal pair for φ. More technically, both $\mathcal{D}_\eta$ and $\mathcal{D}_\rho$ use mn variable components that correspond to the variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively. For every $j \in [m]$, assuming the j-th clause is $\ell_j^1 \vee \ell_j^2 \vee \ell_j^3$, the design $\mathcal{D}_\rho$ must use two additional variable components $\mathcal{B}^{t_1}_{\ell_j^a}$ and $\mathcal{B}^{t_2}_{\ell_j^b}$, for $a \neq b \in \{1, 2, 3\}$ and $t_1, t_2 \in [m]$, which corresponds to η or ρ assigning value true to ℓ_j^a and value false to ℓ_j^b.

Player 1 has an additional correct design $\mathcal{D}_{ALL}$ in which he does not share any components regardless of the other players' choices. Player 2 has two possible designs $\mathcal{D}_A$ and $\mathcal{D}_B$. Assume Player 3 chooses a design $\mathcal{D}_\eta$. We describe the interaction between Player 1 and Player 2. We define the library and the players' objectives so that when Player 1 chooses some design $\mathcal{D}_\rho$, Player 2 prefers $\mathcal{D}_B$ over $\mathcal{D}_A$, thus $cost_2(\langle \mathcal{D}_\rho, \mathcal{D}_A, \mathcal{D}_\eta \rangle) > cost_2(\langle \mathcal{D}_\rho, \mathcal{D}_B, \mathcal{D}_\eta \rangle)$. When Player 2 plays $\mathcal{D}_B$, Player 1 prefers $\mathcal{D}_{ALL}$ over every design $\mathcal{D}_\rho$, thus $cost_1(\langle \mathcal{D}_\rho, \mathcal{D}_B, \mathcal{D}_\eta \rangle) > cost_1(\langle \mathcal{D}_{ALL}, \mathcal{D}_B, \mathcal{D}_\eta \rangle)$. When Player 1 chooses $\mathcal{D}_{ALL}$, Player 2 prefers $\mathcal{D}_A$ over $\mathcal{D}_B$, thus $cost_2(\langle \mathcal{D}_{ALL}, \mathcal{D}_B, \mathcal{D}_\eta \rangle) > cost_2(\langle \mathcal{D}_\eta, \mathcal{D}_A, \mathcal{D}_\eta \rangle)$. Finally, when Player 2 chooses $\mathcal{D}_A$, Player 1 prefers the design $\mathcal{D}_\rho$ iff the pair $\langle \eta, \rho \rangle$ is legal for φ, thus $cost_1(\langle \mathcal{D}_{ALL}, \mathcal{D}_A, \mathcal{D}_\eta \rangle) > cost_1(\langle \mathcal{D}_\rho, \mathcal{D}_A, \mathcal{D}_\eta \rangle)$, for a legal pair $\langle \eta, \rho \rangle$.

Thus, if $\varphi \in$ NAE, then for every assignment η, there is an assignment ρ such that $\langle \eta, \rho \rangle$ is a legal pair. Then, assuming Player 3 chooses a design $\mathcal{D}_\eta$, Player 1 prefers either choosing $\mathcal{D}_{ALL}$ or $\mathcal{D}_\rho$ over every other design, where $\langle \eta, \rho \rangle$ is a legal pair. By the above, there is no PNE in the game. If $\varphi \notin$ NAE, then there is an assignment η such that for every assignment ρ, the pair $\langle \eta, \rho \rangle$ is illegal. Then, the profile $\langle \mathcal{D}_{ALL}, \mathcal{D}_A, \mathcal{D}_\eta \rangle$ is a PNE, and we are done. ◀

References

1 H. Ackermann and A. Skopalik. Complexity of pure Nash equilibria in player-specific network congestion games. *Internet Mathematics*, 5(4):321–515, 2008.

2 S. Aland, D. Dumrauf, M. Gairing, B. Monien, and F. Schoppmann. Exact price of anarchy for polynomial congestion games. *SIAM J. Comput.*, 40(5):1211–1233, 2011.

3 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.

4 G. Avni and O. Kupferman. Synthesis from component libraries with costs. In *Proc. 25th CONCUR*, LNCS 8704, pages 156–172. Springer, 2014.

5 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. In *Proc. 17th FoSSaCS*, LNCS 8412, pages 119–133. Springer, 2014.

6 B. Awerbuch, Y. Azar, and A. Epstein. The price of routing unsplittable flow. *SIAM J. Comput.*, 42(1):160–177, 2013.

7 N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proc. 8th AAMAS*, 2009.

8 K. Bhawalkar, M. Gairing, and T. Roughgarden. Weighted congestion games: Price of anarchy, universal worst-case examples, and tightness. In *ESA (2)*, pages 17–28, 2010.

9 V. Bilò. A unifying tool for bounding the quality of non-cooperative solutions in weighted congestion games. In *WAOA*, pages 215–228, 2012.

10 I. Caragiannis, M. Flammini, C. Kaklamanis, P. Kanellopoulos, and L. Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, 2011.

11 G. Christodoulou and M. Gairing. Price of stability in polynomial congestion games. In *Proc. 40th ICALP*, pages 496–507, 2013.

12 G. Christodoulou and E. Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *ESA*, pages 59–70, 2005.

13 N. Daniele, F. Guinchiglia, and M.Y. Vardi. Improved automata generation for linear temporal logic. In *Proc. 11th CAV*, LNCS 1633, pages 249–260. Springer, 1999.

14 J. Dunkel and A.S. Schulz. On the complexity of pure-strategy nash equilibria in congestion and local-effect games. *Mathematics of Operations Research*, 33(4):851–868, 2008.

15 C. Dwork and M. Naor. Pricing via processing, or, combatting junk mail. In *Proc. CRYPTO*, pages 139–177, 2009.

16 T. Eiter and G. Gottlob Note on the complexity of some eigenvector problems. Technical Report CD-TR 95/89, Christian Doppler Laboratory for Expert Systems, TU Vienna, 1995.

17 D. Fotakis, S. Kontogiannis, and P. Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2-3):226–239, 2005.

18 D. Fotakis, S. Kontogiannis, and P. Spirakis. Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost. In *Proc. WAOA*, pages 161-175, 2005.

19 A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *Proc. 36th STOC*, pages 604–612, 2004.

20 M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. Freeman and Co., 1979.

21 T. Harks and M. Klimm. On the existence of pure Nash equilibria in weighted congestion games. *Math. Oper. Res.*, 37(3):419–436, 2012.

22 E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

23 Y. Lustig and M.Y. Vardi. Synthesis from component libraries. *STTT*, 15(5-6):603–618, 2013.

24 C. Meyers. *Network flow problems and congestion games: complexity and approximation results*. PhD thesis, MIT, 2006.

25 I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13(1):111–124, 1996.

26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

27 M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *IEEE J. on Selected Areas in Communication*, 1998. Issue on Copyright and Privacy Protection.

28 R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

29 T. Roughgarden and E. Tardos. How bad is selfish routing? *JACM*, 49(2):236–259, 2002.

30 L. Tran-Thanh, M. Polukarov, A. C. Chapman, A. Rogers, and N. R. Jennings. On the existence of pure strategy nash equilibria in integer-splittable weighted congestion games. In *SAGT*, pages 236–253, 2011.

The Sensing Cost of Monitoring and Synthesis

Shaull Almagor[1], Denis Kuperberg[2], and Orna Kupferman[1]

1 **The Hebrew University, Israel**
2 **IRIT/Onera, Toulouse, France**

Abstract

In [2], we introduced *sensing* as a new complexity measure for the complexity of regular languages. Intuitively, the sensing cost quantifies the detail in which a random input word has to be read by a deterministic automaton in order to decide its membership in the language. In this paper, we consider sensing in two principal applications of deterministic automata. The first is *monitoring*: we are given a computation in an on-line manner, and we have to decide whether it satisfies the specification. The second is *synthesis*: we are given a sequence of inputs in an on-line manner and we have to generate a sequence of outputs so that the resulting computation satisfies the specification. In the first, our goal is to design a monitor that handles all computations and minimizes the expected average number of sensors used in the monitoring process. In the second, our goal is to design a transducer that realizes the specification for all input sequences and minimizes the expected average number of sensors used for reading the inputs.

We argue that the two applications require new and different frameworks for reasoning about sensing, and develop such frameworks. We focus on safety languages. We show that for monitoring, minimal sensing is attained by a monitor based on the minimal deterministic automaton for the language. For synthesis, however, the setting is more challenging: minimizing the sensing may require exponentially bigger transducers, and the problem of synthesizing a minimally-sensing transducer is EXPTIME-complete even for safety specifications given by deterministic automata.

1998 ACM Subject Classification F.4.3 Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Automata, regular languages, ω-regular languages, complexity, sensing, minimization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.380

1 Introduction

Studying the complexity of a formal language, there are several complexity measures to consider. When the language is given by means of a Turing Machine, the traditional measures are time and space requirements. Theoretical interest as well as practical considerations have motivated additional measures, such as randomness (the number of random bits required for the execution) [11] or communication complexity (number and length of messages required) [10]. For ω-regular languages, given by means of finite-state automata, the classical complexity measure is the size of a minimal deterministic automaton that recognizes the language.

In [2], we introduced and studied a new complexity measure, namely the *sensing cost* of the language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language. Sensing has been studied in several other CS contexts. In theoretical CS, in methodologies such as PCP and property testing, we are allowed to sample or query only part of the input [8]. In more practical applications, mathematical tools in signal processing are used to reconstruct information based on compressed sensing [3], and in the context of data streaming, one

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 380–393

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

cannot store in memory the entire input, and therefore has to approximate its properties according to partial "sketches" [12].

Our study in [2] considered regular and ω-regular languages, where sensing is defined as follows. Consider a deterministic automaton $\mathcal{A}$ over an alphabet 2^P, for a finite set P of *signals*. For a state q of $\mathcal{A}$, we say that a signal $p \in P$ is *sensed* in q if at least one transition taken from q depends on the truth value of p. The *sensing cost* of q is the number of signals it senses, and the sensing cost of a run is the average sensing cost of states visited along the run. We extend the definition to automata by assuming a given distribution of the inputs. The sensing cost of a language with respect to this distribution is then the infimum sensing cost of an automaton for the language. For simplicity, we focus on the uniform distribution, and we refer to the sensing cost of an automaton without parameterizing it by a distribution. As detailed in Remark 1, all our results can be extended to a setting with a parameterized distribution.

In [2], we showed that computing the sensing cost of a language can be done in polynomial time. We further showed that while in finite words the minimal sensing cost is always attained, this is not the case for infinite words. For example, recognizing the language L over $2^{\{p\}}$ of all words with infinitely many p's, one can give up sensing of p for unboundedly-long intervals, thus the sensing cost of L is 0, yet every deterministic automaton $\mathcal{A}$ that recognizes L must sense p infinitely often, causing the sensing cost of $\mathcal{A}$ to be strictly greater than 0.

In the context of formal methods, sensing has two appealing applications. The first is *monitoring*: we are given a computation and we have to decide whether it satisfies a specification. When the computations are over 2^P, we want to design a monitor that minimizes the expected average number of sensors used in the monitoring process. Monitoring is especially useful when reasoning about *safety* specifications [7]. There, every computation that violates the specification has a bad prefix – one all whose extensions are not in L. Hence, as long as the computation is a prefix of some word in L, the monitor continues to sense and examine the computation. Once a bad prefix is detected, the monitor declares an error and no further sensing is required. The second application is *synthesis*. Here, the set P of signals is partitioned into sets I and O of input and output signals, respectively. We are given a specification L over the alphabet $2^{I \cup O}$, and our goal is to construct an I/O *transducer* that realizes L. That is, for every sequence of assignments to the input signals, the transducer generates a sequence of assignments to the output signals so that the obtained computation is in L [13]. Our goal is to construct a transducer that minimizes the expected average number of sensors (of input signals) that are used along the interaction.

The definition of sensing cost in [2] falls short in the above two applications. For the first, the definition in [2] does not distinguish between words in the language and words not in the language, whereas in monitoring we care only for words in the language. In particular, according to the definition in [2], the sensing cost of a safety language is always 0. For the second, the definition in [2] considers automata and does not partition P into I and O, whereas synthesis refers to I/O-transducers. Moreover, unlike automata, correct transducers generate only computations in the language, and they need not generate all words in the language – only these that ensure receptiveness with respect to all sequences of inputs.

In this work we study sensing in the context of monitoring and synthesis. We suggest definitions that capture the intuition of "required number of sensors" in these settings and solve the problems of generating monitors and transducers that minimize sensing. For both settings, we focus on safety languages.

Consider, for example, a traffic monitor that has access to various sensors on roads and whose goal is to detect accidents. Once a road accident is detected, an alarm is raised to the proper authorities and the monitoring is stopped until the accident has been taken care

of. The monitor can read the speed of cars along the roads, as well as the state of traffic lights. An accident is detected when some cars do not move even-though no traffic light is stopping them. Sensing the speed of every car and checking every traffic light requires huge sensing. Our goal is to find a monitor that minimizes the required sensing and still detects all accidents. In the synthesis setting, our goal is extended to designing a transducer that controls the traffic lights according to the speed of the traffic in each direction, and satisfies some specification (say, give priority to slow traffic), while minimizing the sensing of cars.

We can now describe our model and results. Let us start with monitoring. Recall that the definition of sensing in [2] assumes a uniform probability on the assignments to the signals, whereas in monitoring we want to consider instead more intricate probability spaces – ones that restrict attention to words in the language. As we show, there is more than one way to define such probability spaces, each leading to a different measure. We study two such measures. In the first, we sample a word randomly, letter by letter, according to a given distribution, allowing only letters that do not generate bad prefixes. In the second, we construct a sample space directly on the words in the language. We show that in both definitions, we can compute the sensing cost of the language in polynomial time, and that the minimal sensing cost is attained by a minimal-size automaton. Thus, luckily enough, even though different ways in which a computation may be given in an online manner calls for two definitions of sensing cost, the design of a minimally-sensing monitor is the same in the two definitions.

Next, we proceed to study sensing for synthesis. The main challenge there is that we no longer need to consider all words in the language. Also, giving up sensing has a flavor of synthesis with incomplete information [9]: the transducer has to realize the specification no matter what the incomplete information is. This introduces a new degree of freedom, which requires different techniques than those used in [2]. In particular, while a minimal-size transducer for a safety language can be defined on top of the state space of a minimal-size deterministic automaton for the language, this is not the case when we seek minimally-sensing transducers. This is different also from the results in [2] and even these in the monitoring setting, where a minimally-sensing automaton or monitor for a safety language coincides with the minimal-size automaton for it. In fact, we show that a minimally-sensing transducer for a safety language might be exponentially bigger than a minimal-size automaton for the language. Consequently, the problems of computing the minimal sensing cost and finding a minimally-sensing transducer are EXPTIME-complete even for specifications given by means of deterministic safety automata. On the positive side, a transducer that attains the minimal sensing cost always exists for safety specifications. For general ω-regular specifications, even decidability of computing the optimal sensing cost remains open.

Due to lack of space, some of the proofs are omitted and can be found in the full version, in the authors' home pages.

2 Preliminaries

Automata and Transducers

A *deterministic automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \nrightarrow Q$ is a partial transition function, and α is an acceptance condition. We sometimes refer to δ as a relation $\Delta \subseteq Q \times \Sigma \times Q$, with $\langle q, \sigma, q' \rangle \in \Delta$ iff $\delta(q, \sigma) = q'$. A run of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is a sequence of states $q_0, q_1, \ldots$ such that $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for all $i \geq 0$. Note that since δ is deterministic and partial, $\mathcal{A}$ has at most one run on a word. A run is accepting if it satisfies the acceptance

condition. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We denote by $\mathcal{A}^q$ the automaton $\mathcal{A}$ with the initial state set to q.

In a deterministic *looping* automaton (DLW), every run is accepting. Thus, a word is accepted if there is a run of the automaton on it.[1] Since every run is accepting, we omit the acceptance condition and write $\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$.

For finite sets I and O of input and output signals, respectively, an I/O *transducer* is $\mathcal{T} = \langle I, O, Q, q_0, \delta, \rho \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times 2^I \to Q$ is a total transition function, and $\rho : Q \to 2^O$ is a labeling function on the states. The run of $\mathcal{T}$ on a word $w = i_0 \cdot i_1 \cdots \in (2^I)^\omega$ is the sequence of states $q_0, q_1, \ldots$ such that $q_{k+1} = \delta(q_k, i_k)$ for all $k \geq 0$. The *output* of $\mathcal{T}$ on w is then $o_1, o_2, \ldots \in (2^O)^\omega$ where $o_k = \rho(q_k)$ for all $k \geq 1$. Note that the first output assignment is that of q_1, and we do not consider $\rho(q_0)$. This reflects the fact that the environment initiates the interaction. The *computation of* $\mathcal{T}$ *on* w is then $\mathcal{T}(w) = i_0 \cup o_1, i_1 \cup o_2, \ldots \in (2^{I \cup O})^\omega$.

Note that the structure of each I/O-transducer $\mathcal{T}$ induces a DLW $\mathcal{A}_\mathcal{T}$ over the alphabet 2^I with a total transition relation. Thus, the language of the DLW is $(2^I)^\omega$, reflecting the receptiveness of $\mathcal{T}$.

Safety Languages

Consider a language $L \subseteq \Sigma^\omega$. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if for every $y \in \Sigma^\omega$, we have that $x \cdot y \notin L$. That is, x is a bad prefix if all its extensions are words not in L. The language L is then a *safety* language if every word not in L has a bad prefix. For a language L, let $\mathit{pref}(L) = \{x \in \Sigma^* : \text{ there exists } y \in \Sigma^\omega \text{ such that } x \cdot y \in L\}$ be the set of prefixes of words in L. Note that each word in Σ^* is either in $\mathit{pref}(L)$ or is a bad prefix for L. Since the set $\mathit{pref}(L)$ for a safety language L is *fusion closed* (that is, a word is in L iff all its prefixes are in $\mathit{pref}(L)$), an ω-regular language is safety iff it can be recognized by a DLW [15].

Consider a safety language L over sets I and O of input and output signals. We say that L is I/O*-realizable* if there exists an I/O transducer $\mathcal{T}$ all whose computations are in L. Thus, for every $w \in (2^I)^\omega$, we have that $\mathcal{T}(w) \in L$. We then say that $\mathcal{T}$ I/O*-realizes* L. When I and O are clear from the context, we omit them. The *synthesis* problem gets as input a safety language L over $I \cup O$, say by means of a DLW, and returns an I/O-transducer that realizes L or declares that L is not I/O-realizable.

Sensing

In [2], we defined regular sensing as a measure for the number of sensors that need to be operated in order to recognize a regular language. We study languages over an alphabet $\Sigma = 2^P$, for a finite set P of signals. A letter $\sigma \in \Sigma$ corresponds to a truth assignment to the signals, and sensing a signal amounts to knowing its assignment. Describing sets of letters in Σ, it is convenient to use Boolean assertions over P. For example, when $P = \{a, b\}$, the assertion $\neg b$ stands for the set $\{\emptyset, \{a\}\}$ of two letters.

For completeness, we bring here the definitions from [2]. Consider a language L and a deterministic automaton $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$ such that $L(\mathcal{A}) = L$. We assume that δ is total. For a state $q \in Q$ and a signal $p \in P$, we say that p is *sensed in* q if there exists a set $S \subseteq P$ such that $\delta(q, S \setminus \{p\}) \neq \delta(q, S \cup \{p\})$. Intuitively, a signal is sensed in q if knowing its value

[1] For readers familiar with the Büchi acceptance condition, a looping automaton is a special case of Büchi with $\alpha = Q$.

may affect the destination of at least one transition from q. We use $sensed(q)$ to denote the set of signals sensed in q. The *sensing cost* of a state $q \in Q$ is $scost(q) = |sensed(q)|$. [2]

For a finite run $r = q_1, \ldots, q_m$ of $\mathcal{A}$, we define the sensing cost of r, denoted $scost(r)$, as $\frac{1}{m}\sum_{i=0}^{m-1} scost(q_i)$. That is, $scost(r)$ is the average number of sensors that $\mathcal{A}$ uses during r. Now, for a finite word w, we define the sensing cost of w in $\mathcal{A}$, denoted $scost_{\mathcal{A}}(w)$, as the sensing cost of the run of $\mathcal{A}$ on w. Finally, the sensing cost of $\mathcal{A}$ is the expected sensing cost of words of length that tends to infinity, where we assume that the letters in Σ are uniformly distributed (see Remark 1 below). Thus, $scost(\mathcal{A}) = \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w\in\Sigma^m} scost_{\mathcal{A}}(w)$.

Note that the definition applies to automata on both finite and infinite words, and it corresponds to the closed setting: the automaton gets as input words over 2^P and uses sensors in order to monitor the input words and decide their membership in L. We define the *sensing cost* of a language L to be the minimal cost of an automaton for L. A-priori, the minimal cost might not be attained by a single automaton, thus we define $scost(L) = \inf\{scost(\mathcal{A}) : \mathcal{A} \text{ is an automaton for } L\}$.

▶ **Remark 1** (On the choice of uniform distribution). *The choice of a uniform distribution on the letters in Σ may be unrealistic in practice. Indeed, in real scenarios, the distribution on the truth assignments to the underlying signals may be complicated. Generally, such a distribution can be given by a Markov chain (in monitoring) or by an MDP (in synthesis). As it turns out, adjusting our setting and algorithms to handle such distributions involves only a small technical elaboration, orthogonal to the technical challenges that exists already in a uniform distribution.*

Accordingly, throughout the paper we assume a uniform distribution on the truth assignments to the signals. In the full version we describe how our setting and algorithms are extended to the general case.

The definition of sensing in [2] essentially considers the sensing required in the Ergodic SCC of a deterministic automaton for the language. Since in safety languages, the Ergodic SCCs are accepting or rejecting sinks, which require no sensing, we have the following, which implies that the definition in [2] is not too informative for safety languages.

▶ **Lemma 2.** *For every safety language $L \subseteq \Sigma^\omega$, we have $scost(L) = 0$.*

Markov Chains and Decision Processes

A Markov chain $\mathcal{M} = \langle S, P\rangle$ consists of a finite state space S and a stochastic transition matrix $P : S \times S \to [0,1]$. That is, for all $s \in S$, we have $\sum_{s'\in S} P(s,s') = 1$. Given an initial state s_0, consider the vector v^0 in which $v^0(s_0) = 1$ and $v^0(s) = 0$ for every $s \neq s_0$. The *limiting distribution* of $\mathcal{M}$ is $\lim_{n\to\infty} \frac{1}{n}\sum_{m=0}^{n} v^0 P^m$. The limiting distribution satisfies $\pi P = \pi$, and can be computed in polynomial time [5].

A Markov decision process (MDP) is $\mathcal{M} = \langle S, s_0, (A_s)_{s\in S}, \mathrm{P}, cost\rangle$ where S is a finite set of states, $s_0 \in S$ is an initial state, A_s is a finite set of actions that are available in state $s \in S$. Let $A = \bigcup_{s\in S} A_s$. Then, $\mathrm{P} : S \times A \times S \nrightarrow [0,1]$ is a partial transition probability function, defining for every two states $s, s' \in S$ and action $a \in A_s$, the probability of moving from s to s' when action a is taken. Accordingly, $\sum_{s'\in S} \mathrm{P}(s,a,s') = 1$. Finally, $cost : S \times A \nrightarrow \mathbb{N}$ is a

[2] We note that, alternatively, one could define the *sensing level* of states, with $slevel(q) = \frac{scost(q)}{|P|}$. Then, for all states q, we have that $slevel(q) \in [0,1]$. All our results hold also for this definition, simply by dividing the sensing cost by $|P|$.

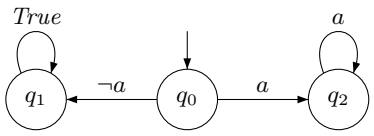


Figure 1 A DLW for $a^\omega + (\neg a) \cdot (True)^\omega$.

partial cost function, assigning each state s and action $a \in A_s$, the cost of taking action a in state s.

An MDP can be thought of as a game between a player who chooses the actions and nature, which acts stochastically according to the transition probabilities.

A *policy* for an MDP $\mathcal{M}$ is a function $f : S^* \times S \to A$ that outputs an action given the history of the states, such that for $s_0, \ldots, s_n$ we have $f(s_0, \ldots, s_n) \in A_{s_n}$. Policies correspond to the strategies of the player. The *cost* of a policy f is the expected average cost of a random walk in $\mathcal{M}$ in which the player proceeds according to f. Formally, for $m \in \mathbb{N}$ and for a sequence of states $\tau = s_0, \ldots, s_{m-1}$, we define $\mathrm{P}_f(\tau) = \prod_{i=1}^{m-1} \mathrm{P}(s_{i-1}, f(s_0 \cdots s_{i-1}), s_i)$. Then, $cost_m(f, \tau) = \frac{1}{m} \sum_{i=1}^{m} cost(s_i, f(s_1 \cdots s_i))$ and we define the cost of f as $cost(f) = \liminf_{m\to\infty} \frac{1}{m} \sum_{\tau:|\tau|=m} cost_m(f, \tau) \cdot \mathrm{P}_f(\tau)$.

A policy is *memoryless* if it depends only on the current state. We can describe a memoryless policy by $f : S \to A$. A memoryless policy f induces a Markov chain $\mathcal{M}^f = \langle S, P_f \rangle$ with $P_f(s, s') = \mathrm{P}(s, f(s), s')$. Let π be the limiting distribution of $\mathcal{M}^f$. It is not hard to prove that $cost(f) = \sum_{s\in S} \pi_s cost(s, f(s))$. Let $cost(\mathcal{M}) = \inf\{cost(f) : f \text{ is a policy for } \mathcal{M}\}$. That is, $cost(M)$ is the expected cost of a game played on $\mathcal{M}$ in which the player uses an optimal policy.

▶ **Theorem 3.** *Consider an MDP $\mathcal{M}$. Then, $cost(\mathcal{M})$ can be attained by a memoryless policy, which can be computed in polynomial time.*

3 Monitoring

As described in Section 2, the definition of sensing in [2] takes into an account all words in $(2^P)^\omega$, regardless their membership in the language. In monitoring, we restrict attention to words in the language, as once a violation is detected, no further sensing is required. In particular, in safety languages, violation amounts to a detection of a bad prefix, and indeed safety languages are the prominent class of languages for which monitoring is used [7].

As it turns out, however, there are many approaches to define the corresponding probability space. We suggest here two. Let $\mathcal{A}$ be a DLW and let $L = L(\mathcal{A})$.

1. **[Letter-based]** At each step, we uniformly draw a "safe" letter – one with which we are still generating a word in $pref(L)$, thereby iteratively generating a random word in L.
2. **[Word-based]** At the beginning, we uniformly draw a word in L.

We denote the sensing cost of $\mathcal{A}$ in the letter- and word-based approaches $lcost(\mathcal{A})$ and $wcost(\mathcal{A})$, respectively. The two definitions yield two different probability measures on L, as demonstrated in Example 4 below.

▶ **Example 4.** Let $P = \{a\}$ and consider the safety language $L = a^\omega + (\neg a) \cdot (True)^\omega$. That is, if the first letter is $\{a\}$, then the suffix should be $\{a\}^\omega$, and if the first letter is $\emptyset$, then all suffixes result in a word in L. Consider the DLW $\mathcal{A}$ for L in Figure 1.

In the letter-based definition, we initially draw a letter from $2^{\{a\}}$ uniformly, i.e., either a or $\neg a$ w.p. $\frac{1}{2}$. If we draw $\neg a$, then we move to q_1 and stay there forever. If we draw a, then

we move to q_2 and stay there forever. Since $scost(q_1) = 0$ and $scost(q_2) = 1$, and we reach q_1 and q_2 w.p $\frac{1}{2}$, we get $lcost(\mathcal{A}) = \frac{1}{2}$.

In the word-based definition, we assign a uniform probability to the words in L. In this case, almost all words are not a^ω, and thus the probability of a^ω is 0. This means that we will get to q_1 w.p. 1, and thus $wcost(\mathcal{A}) = 0$.

As a more realistic example, recall our traffic monitor in Section 1. There, the behavior of the cars is the random input, and the two approaches can be understood as follows. In the letter-based approach, we assume that the drivers do their best to avoid accidents regardless of the history of the traffic and the traffic lights so far. Thus, after every safe prefix, we assume that the next input is also safe. In the word-based approach, we assume that the city is planned well enough to avoid accidents. Thus, we a-priori set the distribution to safe traffic behaviors according to their likelihood.

We now define the two approaches formally.

The Letter-Based Approach

Consider a DLW $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$. For a state $q \in Q$, let $avail(q)$ be the set of letters available in q, namely letters that do not cause $\mathcal{A}$ to get stuck. Formally, $avail(q) = \{\sigma \in \Sigma : \delta(q, \sigma) \text{ is defined } \}$. We model the drawing of available letters by the Markov chain $\mathcal{M}_\mathcal{A} = \langle Q, P \rangle$, where the probability of a transition from state q to state q' in $\mathcal{M}_\mathcal{A}$ is $P(q, q') = \frac{|\{\sigma \in \Sigma : \delta(q,\sigma) = q'\}|}{|avail(q)|}$. Let π be the limiting distribution of $\mathcal{M}_\mathcal{A}$. We define $lcost(\mathcal{A}) = \sum_{q \in Q} \pi(q) \cdot scost(q)$.

Since computing the limiting distribution can be done in polynomial time, we have the following.

▶ **Theorem 5.** *Given a* DLW $\mathcal{A}$*, the sensing cost* $lcost(\mathcal{A})$ *can be calculated in polynomial time.*

The Word-Based Approach

Consider a DLW $\mathcal{A} = \langle 2^P, Q, q_0, \delta \rangle$ recognizing a non-empty safety language L. From [2], we have $scost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{|\Sigma|^n} \sum_{u \in \Sigma^n} scost_\mathcal{A}(u)$, which coincides with $\mathbb{E}[scost_\mathcal{A}(u)]$ where $\mathbb{E}$ is the expectation with respect to the standard measure on Σ^ω. Our goal here is to replace this standard measure with one that restricts attention to words in L. Thus, we define $wcost(\mathcal{A}) = \mathbb{E}[scost(u) \mid u \in L]$. For $n \geq 0$, let $pref(L, n)$ be the set of prefixes of L of length n. Formally, $pref(L, n) = pref(L) \cap \Sigma^n$. As in the case of the standard measure, the expectation-based definition coincides with one that that is based on a limiting process: $wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{|pref(L,n)|} \sum_{u \in pref(L,n)} scost_\mathcal{A}(u)$. Thus, the expressions for $scost$ and $wcost$ are similar, except that in the expectation-based definition we add conditional probability, restricting attention to words in L, and in the limiting process we replace Σ^n by $pref(L, n)$.

Note that the term $\frac{1}{|pref(L,n)|}$ is always defined, as L is a non-empty safety language. In particular, the expectation is well defined even if L has measure 0 in Σ^ω.

▶ **Theorem 6.** *Given a DLW* $\mathcal{A}$*, we can compute* $wcost(\mathcal{A})$ *in polynomial time.*

Proof. We will use here formal power series on one variable z, a classical tool for graph and automata combinatorics. They can be thought of as polynomials of infinite degree.

For states $p, q \in Q$ and for $n \in \mathbb{N}$, let $\#paths(p, q, n)$ denote the number of paths (each one labeled by a distinct word) of length n from p to q in $\mathcal{A}$. We define the generating functions:

$C_{p,q}(z) = \sum_{n\in\mathbb{N}} \#paths(p,q,n)z^n$ and $F_q(z) = C_{q_0,q}(z)\sum_{p\in Q} C_{q,p}(z)$. Let $[z^n]F_q(z)$ be the coefficient of z^n in $F_q(z)$. By the definition of $C_{q_0,q}$, we get

$$[z^n]F_q(z) = \sum_{k=0}^{n} \#paths(q_0,q,k) \sum_{p\in Q} \#paths(q,p,n-k).$$

Therefore, $[z^n]F_q(z)$ is the total number of times the state q is used when listing all paths of length n from q_0.

Thus, we have $\sum_{u\in pref(L,n)} scost(u) = \frac{1}{n}\sum_{q\in Q} scost(q)[z^n]F_q(z)$. Finally, let $S(z) = \sum_{p\in p} C_{q_0,p}(z)$. Then, $wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{n\cdot[z^n]S(z)} \sum_{q\in Q} scost(q)[z^n]F_q(z)$. In the full version we use techniques from [4] and [14] to compute the latter limit in polynomial time, by asymptotic estimations of the coefficients, thus concluding the proof. ◀

Sensing cost of languages

For a safety language L, we define $lcost(L) = \inf\{lcost(\mathcal{A}) : \mathcal{A} \text{ is a DLW for } L\}$, and similarly for $wcost(L)$. Different DLWs for a language L may have different sensing costs. We show that the minimal sensing cost in both approaches is attained at the minimal-size DLW. We first need some definitions and notations.

Consider a safety language $L \subseteq \Sigma^\omega$. For two finite words u_1 and u_2, we say that u_1 and u_2 are *right L-indistinguishable*, denoted $u_1 \sim_L u_2$, if for every $z \in \Sigma^\omega$, we have that $u_1 \cdot z \in L$ iff $u_2 \cdot z \in L$. Thus, $\sim_L$ is the Myhill-Nerode right congruence used for minimizing DFAs. For $u \in \Sigma^*$, let $[u]$ denote the equivalence class of u in $\sim_L$ and let $\langle L\rangle$ denote the set of all equivalence classes. Each class $[u] \in \langle L\rangle$ is associated with the *residual language* $u^{-1}L = \{w : uw \in L\}$. Note that for safety languages, there is at most one class $[u]$, namely the class of bad prefixes, such that $u^{-1}L = \emptyset$. We denote this class $[\bot]$. When $L \neq \emptyset$ is a regular safety language, the set $\langle L\rangle$ is finite, and induces the *residual automaton* of L, defined by $\mathcal{R}_L = \langle \Sigma, \langle L\rangle \setminus \{[\bot]\}, \delta_L, [\epsilon]\rangle$, with $\delta_L([u],a) = [u\cdot a]$ for all $[u] \in \langle L\rangle \setminus \{[\bot]\}$ and $a \in \Sigma$ such that $[u\cdot a] \neq [\bot]$. The automaton $\mathcal{R}_L$ is well defined and is the unique minimal-size DLW for L.

Consider a DLW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta\rangle$ such that $L(\mathcal{A}) = L$. For a state $s = [u] \in \langle L\rangle \setminus \{[\bot]\}$, we associate with s a set $states(\mathcal{A}, s) = \{q \in Q : L(\mathcal{A}^q) = u^{-1}L\}$. That is, $states(\mathcal{A}, s) \subseteq Q$ contains exactly all state that $\mathcal{A}$ can be in after reading a word that leads $\mathcal{R}_L$ to $[u]$.

The following claims are simple exercises.

▶ **Proposition 7.** *Consider a safety language L and a DLW $\mathcal{A}$ for it.*

1. *The set $\{states(\mathcal{A}, s) : s \in \langle L\rangle \setminus \{[\bot]\}\}$ forms a partition of the states of $\mathcal{A}$.*
2. *For every state $s \in \langle L\rangle \setminus \{[\bot]\}$ of $\mathcal{R}_L$, letter $\sigma \in \Sigma$, and state $q \in states(\mathcal{A}, s)$, we have $\delta(q,\sigma) \in states(\mathcal{A}, \delta_L(s,\sigma))$.*

▶ **Lemma 8.** *Consider a safety language $L \subseteq \Sigma^\omega$. For every* DLW *$\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $lcost(\mathcal{A}) \geq lcost(\mathcal{R}_L)$ and $wcost(\mathcal{A}) \geq wcost(\mathcal{R}_L)$*

Proof. We outline the key points in the proof for *lcost*. The arguments for *wcost* are similar. For a detailed proof see the full version.

Recall that the states of $\mathcal{R}_L$ are $\langle L\rangle \setminus \{[\bot]\}$. We start by showing that for every $s \in \langle L\rangle \setminus \{[\bot]\}$ and for every $q \in states(\mathcal{A}, s)$ we have that $scost(q) \geq scost(s)$. Next, we consider the Markov chains $\mathcal{M}_\mathcal{A}$ and $\mathcal{M}_{\mathcal{R}_L}$. Using Proposition 7 we show that if π and τ are the limiting distributions of $\mathcal{M}_\mathcal{A}$ and $\mathcal{M}_{\mathcal{R}_L}$ respectively, then for every $s \in \langle L\rangle \setminus \{[\bot]\}$ we have that $\tau(s) = \sum_{q\in states(\mathcal{A},s)} \pi(q)$. Finally, since Q is partitioned by $\{states(\mathcal{A}, s)\}_s$ we conclude that $lcost(\mathcal{A}) \geq lcost(\mathcal{R}_L)$. ◀

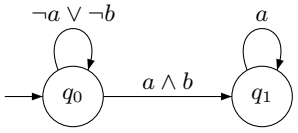


Figure 2 A DLW for $(\neg a \vee \neg b)^\omega + (\neg a \vee \neg b)^* \cdot (a \wedge b) \cdot a^\omega$.

Lemma 8 and Theorems 5 and 6 allow us to conclude with the following.

▶ **Theorem 9.** *Given a* DLW $\mathcal{A}$*, we can compute* $lcost(L(\mathcal{A}))$ *and* $wcost(L(\mathcal{A}))$ *in polynomial time.*

▶ **Example 10.** Consider the DLW $\mathcal{A}$ over the alphabet $2^{\{a,b\}}$ appearing in Figure 2.

Clearly, $\mathcal{A}$ is a minimal automaton for $L = (\neg a \vee \neg b)^\omega + (\neg a \vee \neg b)^* \cdot (a \wedge b) \cdot a^\omega$. By Lemma 8, we can calculate the sensing cost of $\mathcal{A}$ in order to find the sensing cost of L.

Clearly, $scost(q_0) = 2$ and $scost(q_1) = 1$. We start by computing $lcost(\mathcal{A})$. The corresponding Markov chain $M_\mathcal{A}$ has only one ergodic component $\{q_1\}$, so we obtain $lcost(\mathcal{A}) = scost(q_1) = 1$. The computation of $wcost(\mathcal{A})$ is more intricate. In the full version we show that $wcost(\mathcal{A}) = 2$. We remark that unlike in the other versions of sensing cost, transient components can play a role in *wcost*. In particular, If the self-loop on q_0 has been labeled by two rather than three letters, then we would have gotten $wcost(\mathcal{A}) = \frac{3}{2}$.

4 Synthesis

In the setting of synthesis, the signals in P are partitioned into sets I and O of input and output signals. An I/O-transducer $\mathcal{T}$ senses only input signals and we define its sensing cost as the sensing cost of the DLW $\mathcal{A}_\mathcal{T}$ it induces.

We define the *I/O-sensing cost* of a realizable specification $L \in (2^{I \cup O})^\omega$ as the minimal cost of an I/O-transducer that realizes L. Thus, $scost_{I/O}(\mathcal{A}) = \inf\{scost(\mathcal{T}) : \mathcal{T}$ is an I/O-transducer that realizes $L\}$. In this section we consider the problem of finding a minimally-sensing I/O-transducer that realizes L.

The realizability problem for DLW specifications can be solved in polynomial time. Indeed, given a DLW $\mathcal{A}$, we can view $\mathcal{A}$ as a game between a system, which controls the outputs, and an environment, which controls the inputs. We look for a strategy for the system that never reaches an undefined transition. This amounts to solving a turn-based safety game, which can be done in polynomial time.

When sensing is introduced, it is not enough for the system to win this game, as it now has to win while minimizing the sensing cost. Intuitively, not sensing some inputs introduces incomplete information to the game: once the system gives up sensing, it may not know the state in which the game is and knows instead only a set of states in which the game may be. In particular, unlike usual realizability, a strategy that minimizes the sensing need not use the state space of the DLW. We start with an example illustrating this.

▶ **Example 11.** Consider the DLW $\mathcal{A}$ appearing in Figure 3. The DLW is over $I = \{p, q\}$ and $O = \{a\}$. A realizing transducer over the structure of $\mathcal{A}$ (see $\mathcal{T}_1$ in Figure 4) senses p and q, responds with a if $p \wedge q$ was sensed and responds with $\neg a$ if $\neg p \wedge \neg q$ was sensed. In case other inputs are sensed, the response is arbitrary (denoted $*$ in the figure). As $\mathcal{T}_1$ demonstrates, every transducer that is based on the structure of $\mathcal{A}$ senses two input signals (both p and q) every second step, thus its sensing cost is 1. As demonstrated by the transducer $\mathcal{T}_2$ in Figure 5, it is possible to realize $\mathcal{A}$ with sensing cost of $\frac{1}{2}$ by only sensing p every second

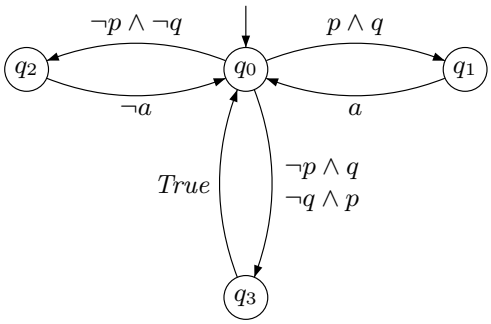


Figure 3 The DLW $\mathcal{A}$ in Example 11.

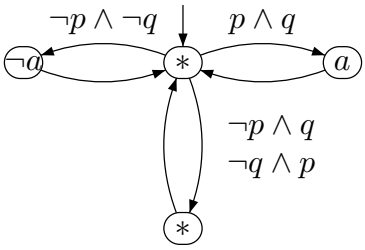


Figure 4 The transducer $\mathcal{T}_1$ for $\mathcal{A}$.

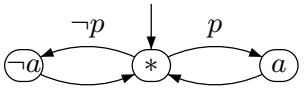


Figure 5 The transducer $\mathcal{T}_2$ for $\mathcal{A}$.

step. Indeed, knowing the value of p is enough in order to determine the output. Note that $\mathcal{T}_2$ may output sometimes a and sometimes $\neg a$ after reading assignments that causes $\mathcal{A}$ to reach q_3. Such a behavior cannot be exhibited by a transducer with the state-structure of $\mathcal{A}$.

Solving games with incomplete information is typically done by some kind of a subset-construction, which involves an exponential blow up. Unlike usual games with incomplete information, here the strategy of the system should not only take care of the realizability but also decides which input signals should be sensed, where the goal is to obtain a minimally sensing transducer. In order to address these multiple objectives, we first construct an MDP in which the possible policies are all winning for the system, and corresponds to different choices of sensing. An optimal policy in this MDP then induces a minimally-sensing transducer.

▶ **Theorem 12.** *Consider a* DLW $\mathcal{A}$ *over* $2^{I \cup O}$. *If* $\mathcal{A}$ *is realizable, then there exists an MDP* $\mathcal{M}$ *in which an optimal strategy corresponds to a minimally-sensing* I/O*-transducer that realizes* $\mathcal{A}$. *The MDP* $\mathcal{M}$ *has size exponential in* $|\mathcal{A}|$ *and can be computed in time exponential in* $|\mathcal{A}|$.

Proof. Consider a DLW $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta \rangle$. We obtain from $\mathcal{A}$ an MDP $\mathcal{M} = \langle \mathcal{S}, \text{START}, A, \text{P}, \mathit{cost} \rangle$, where $\mathcal{S} = (2^Q \times \{0, 1, \bot\}^I) \cup \{\text{START}\}$, and $A = 2^I \times 2^O$. Intuitively, when $\mathcal{M}$ is in state $\langle S, \ell \rangle$, for $S \subseteq Q$ and $\ell : I \to \{0, 1, \bot\}$, then $\mathcal{A}$ can be in every state in S, and for each input signal $b \in I$, we have that either b is true ($\ell(b) = 1$), b is false ($\ell(b) = 0$), or b is not sensed ($\ell(b) = \bot$). The action (o, i) means that we now output o and in the next state we will sense only inputs in i. For $\star \in \{\bot, 0, 1\}$, we define $\ell_\star = \{b \in I : \ell(b) = \star\}$.

We define the actions so that an action $\langle o, i \rangle$ is available in state $\langle S, \ell \rangle$ if for every $q \in S$ and $i' \subseteq \ell_\bot$, we have that $\delta(q, \ell_1 \cup i' \cup o)$ is defined. That is, an action is available if its o component does not cause $\mathcal{A}$ to get stuck no matter what the assignment to the signals that are not sensed is.

The transition probabilities are defined as follows. Consider a state $\langle S, \ell \rangle$, and an available action $\langle o, i \rangle$. Let $S' = \bigcup_{q \in S} \bigcup_{i' \subseteq \ell_\bot} \{\delta(q, \ell_1 \cup i' \cup o)\}$. Recall that by taking action $\langle o, i \rangle$, we decide that in the next state we will only sense signals in i. For $i \subseteq I$, we say that an assignment $\ell' : I \to \{0, 1, \bot\}$ *senses* i if $\ell'_1 \cup \ell'_0 = i$. Note that there are $2^{|i|}$ assignments that sense i. Accordingly, we have $\text{P}(\langle S, \ell \rangle, \langle o, i \rangle, \langle S', \ell' \rangle) = 2^{-|i|}$ for every $\ell' : I \to \{0, 1, \bot\}$ that senses i. That is, a transition from $\langle S, \ell \rangle$ with $\langle o, i \rangle$ goes to the set of all possible successors of S under inputs that are consistent with ℓ and the output assignment o, and the ℓ' component

is selected with uniform distribution among all assignments that sense i. The cost function depends on the number of signals we sense, thus $cost(\langle S, \ell\rangle) = |\ell_1 \cup \ell_0|$.

Finally, in the state START we only choose an initial set of input signals to sense. Thus, for every ℓ such that $\ell_1 \cup \ell_0$, we have $\mathrm{P}(\text{START}, \langle o, i\rangle, \langle\{q_0\}, \ell\rangle) = 2^{-|i|}$. Note that START is not reachable from any state in $\mathcal{M}$, and thus its cost is irrelevant. We arbitrarily set $cost(\text{START}) = 0$.

In the full version we prove that $cost(\mathcal{M}) = scost_{I,O}(\mathcal{A})$ and that a minimal-cost policy f in $\mathcal{M}$ induces a minimally-sensing I/O-transducer that realizes $\mathcal{A}$. Intuitively, we prove this by showing a correspondence between transducers and policies, such that the sensing cost of a transducer $\mathcal{T}$ equals the value of the policy it corresponds to in $\mathcal{M}$.

Finally, we observe that the size of $\mathcal{M}$ is single exponential in the size of $\mathcal{A}$, and that we can construct $\mathcal{M}$ in time exponential in the size of $\mathcal{A}$. ◀

▶ **Theorem 13.** *A minimally-sensing transducer for a realizable* DLW $\mathcal{A}$ *has size tightly exponential in* $|\mathcal{A}|$.

Proof. The upper bound follows from Theorem 3 applied to the MDP constructed in Theorem 12.

For the lower bound, we describe a family of realizable DLWs $\mathcal{A}_1, \mathcal{A}_2, \ldots$ such that for all $k \geq 1$, the DLW $\mathcal{A}_k$ has $1 + \sum_{i=1}^{k} p_i$ states, yet a minimally-sensing transducer for it requires at least $\prod_{i=1}^{k} p_i$ states, where $p_1, p_2, \ldots$ are prime numbers. Intuitively, $\mathcal{A}_k$ is constructed as follows. In the initial state q_{reset}, the inputs signals determine a number $1 \leq i \leq k$, and $\mathcal{A}_k$ moves to component i, which consists of a cycle of length p_i. In every state j in component i, the output signals must acknowledge that $\mathcal{A}_k$ is in state $0 \leq j < p_i$ of component i. Furthermore, we force a sensing of 1 in every state except for q_{reset} by requiring a signal to be acknowledged in every step. Finally, we can go back to q_{reset} only with a special output signal, which can be outputted only in state 0 of an i component.

Thus, a realizing transducer essentially only chooses which signals to read in q_{reset}. We show that 0 bits can be read, but in that case we need $\prod_{i=1}^{k} p_i$ states. Indeed, the transducer needs to keep track of the location in all the i components simultaneously, which means keeping track of the modulo from each p_i. Since every combination of such modulos is possible, the transducer needs $\prod_{i=1}^{k} p_i$ states. In the full version we formalize this intuition. ◀

We now turn to study the complexity of the problem of finding a minimally-sensing transducer. By the construction in Theorem 12 and the polynomial time algorithm from Theorem 3, we have the following.

▶ **Theorem 14.** *Consider a realizable* DLW $\mathcal{A}$ *over* $2^{I \cup O}$. *We can calculate* $cost_{I,O}(\mathcal{A})$ *and return a minimally-sensing* I/O*-transducer that realizes* $\mathcal{A}$ *in time exponential in* $|\mathcal{A}|$.

In order to complete the picture, we consider the corresponding decision problem. Given a DLW $\mathcal{A}$ over $2^{I \cup O}$ and a threshold γ, the sensing problem in the open setting is to decide whether $cost_{I,O}(\mathcal{A}) < \gamma$.

▶ **Theorem 15.** *The sensing problem in the open setting is EXPTIME-complete.*

Proof. The upper bound follows from Theorem 14. For the lower bound, we show that the problem is EXPTIME hard even for a fixed γ. Given a DLW specification $\mathcal{A}$ over $2^{I \cup O}$, we show that it is EXPTIME-hard to decide whether there exists a transducer $\mathcal{T}$ that realizes $\mathcal{A}$ with $scost(\mathcal{T}) < 1$. We show a reduction from the problem of deciding the nonemptiness of an intersection of finite deterministic tree automata proved to be EXPTIME-hard in [6].

The idea is similar to that of Theorem 13, where a reset state is used to select an object, and a transducer can ignore the inputs in this state by using a response which is acceptable in every possible selected object.

A deterministic automaton on finite trees (DFT) is $\mathcal{U} = \langle \Sigma, Q, \delta, q_0, F\rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q \times Q$ is a transition function, and $F \subseteq Q$ is a set of accepting states. We refer to the left and right components of δ as $\delta_\triangleleft$ and $\delta_\triangleright$. For example, when $\delta(q,\sigma) = \langle q_l, q_r\rangle$, we write $\delta_\triangleleft(q,\sigma) = q_l$. An DFT runs on Σ-*trees*. A (binary) Σ-tree is $T = \langle \tau, \ell\rangle$ where $\tau \subseteq \{\triangleleft,\triangleright\}^*$ is prefix-closed: for every $x \cdot \sigma \in \tau$ it holds that $x \in \tau$, and $\ell : \tau \to \Sigma$ is a labeling function. For simplicity, we require that for every $x \in \tau$, either $\{x\triangleleft, x\triangleright\} \subseteq \tau$, or $\{x\triangleleft, x\triangleright\} \cap \tau =$, in which case x is a leaf. Given a tree $T = \langle \tau, \ell\rangle$, the *run* of $\mathcal{U}$ on T is a Q-tree $\langle \tau, \ell'\rangle$ where $\ell'(\epsilon) = q_0$, and for every $x \in \tau$ such that x is not a leaf, we have $\delta(\ell'(x), \ell(x)) = \langle \ell'(x\triangleleft), \ell'(x\triangleright)\rangle$. A run is *accepting* if every leaf is labeled by an accepting state. A Σ-tree T is accepted by $\mathcal{U}$ if the run of $\mathcal{U}$ on T is accepting.

The nonempty-intersection problem gets as input DFTs $\mathcal{U}_1, \ldots, \mathcal{U}_n$, and decides whether their intersection is nonempty, that is $\bigcap_{t=1}^n L(\mathcal{U}_t) \neq \emptyset$. Given $\mathcal{U}_1, \ldots, \mathcal{U}_n$, we construct a specification DLW $\mathcal{A}$ such that $\bigcap_{t=1}^n L(\mathcal{U}_t) \neq \emptyset$ iff $scost(\mathcal{A}) < 1$. We assume w.l.o.g. that $L(\mathcal{U}_t) \neq \emptyset$ for all $1 \leq t \leq n$.

We construct $\mathcal{A}$ as follows. Initially, the inputs specify an index $1 \leq t \leq n$. Then, the transducer should respond with a tree in $L(\mathcal{U}_t)$. This is done by challenging the transducer with a branch in the tree, until some reset input signal is true, and the process repeats. Now, if $\bigcap_{t=1}^n L(\mathcal{U}_t) \neq \emptyset$, the transducer can ignore the input signals that specify the index t and just repeatedly output a tree in the intersection. On the other hand, if $\bigcap_{t=1}^n L(\mathcal{U}_t) = \emptyset$, the transducer must sense some information about the specified index.[3]

We now formalize this intuition. For $1 \leq t \leq n$, let $\mathcal{U}_t = \langle 2^J, Q^t, \delta^t, q_0^t, F^t\rangle$. Note that we assume w.l.o.g that the alphabet of all the DFTs is 2^J. We construct a specification DLW $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta\rangle$ as follows. The set of states of $\mathcal{A}$ is $Q = \bigcup_{t=1}^n Q^t \cup \{\textsc{reset}\}$. Assume w.l.o.g that $n = 2^k$ for some $k \in \mathbb{N}$. We define $I = \{b_1, \ldots, b_k\} \cup \{dI\}$ and $O = J \cup \{dO, e\}$. The input signal dI and the output signal dO denote the direction of branching in the tree. For clarity, in an input letter $i \in I$ we write $i(dI) = \triangleleft$ (and $i(dI) = \triangleright$) to indicate that $dI \notin i$ (and $dI \in i$). We use a similar notation for dO.

We define the transition function as follows. In state $\textsc{reset}$, we view the inputs $b_1, \ldots, b_k$ as a binary encoding of a number $t \in \{1, \ldots, n\}$. Then, $\delta(\textsc{reset}, t) = q_0^t$. Next, consider a state $q \in Q^t$, and consider letters $i \subseteq I$ and $o \subseteq O$. We define δ as follows:

$$\delta(q, i \cup o) = \begin{cases} \textsc{reset} & q \in F \wedge e \in o \wedge o(dO) = i(dI) \\ \delta^t_\triangleleft(q, o \cap J) & e \notin o \wedge o(dO) = i(dI) = \triangleleft \\ \delta^t_\triangleright(q, o \cap J) & e \notin o \wedge o(dO) = i(dI) = \triangleright \end{cases}$$

Note that $\delta(q, i \cup o)$ is undefined when $o(dO) \neq i(dI)$ or when $q \notin F$ and $e \in o$. Intuitively, in state $\textsc{reset}$, an index $1 \leq t \leq n$ is chosen. From then on, in a state $q \in Q^t$, we simulate the run of $\mathcal{U}_t$ on the left or right branch of the tree, depending on the signal dI. The next letter is outputted in o, and additionally, we require that dO matches dI.

We claim that $scost(\mathcal{A}) < 1$ iff $\bigcap_{t=1}^n L(\mathcal{U}_t) \neq \emptyset$. In the first direction, assume that $\bigcap_{t=1}^n L(\mathcal{U}_t) \neq \emptyset$, and let T be a tree such that $T \in \bigcap_{t=1}^n L(\mathcal{U}_t) \neq \emptyset$. Consider the following

[3] Note that since a tree in the intersection of DFTs may be exponentially bigger than the DFTs, the lower bound here also suggests an alternative lower bound to the exponential size of a minimally-sensed transducer, now with a polynomial set of signals (as opposed to the proof of Theorem 13).

transducer $\mathcal{T}$: in the state RESET it does not sense any inputs, and then it outputs a branch of T according to the signal dI, while always acknowledging the dI bit with the correct dO. When the end of the branch is reached, it outputs e. Since T is accepted by every DFT U^t, it follows that $\mathcal{T}$ realizes $\mathcal{A}$. Moreover, let l be the longest branch in T, then every l steps at most, $\mathcal{T}$ visits a state corresponding to RESET, in which it senses nothing. Thus, $\mathcal{T}$ senses 1 for at most l steps, and then 0. It follows that $scost(\mathcal{T}) \leq \frac{l}{l+1} = 1 - \frac{1}{l+1} < 1$.

Conversely, observe that in every state $q \in Q \setminus \{\text{RESET}\}$, a realizing transducer must sense at least 1 signal, namely dI. Thus, the only way to get sensing cost of less than 1 is to visit RESET infinitely often (in fact, with bounded sparsity), and to sense 0 in RESET. However, sensing 0 in RESET means that the next state could be the initial state of any of the n DFTs. Moreover, visiting RESET again means that at some point e was outputted in an accepting state of one of the DFTs. Thus, the transducer outputs a tree that is accepted in every DFT, so $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$.

Finally, observe that the reduction is clearly polynomial, and thus we conclude that deciding whether $scost(\mathcal{A}) < 1$ is EXPTIME-hard. ◀

5 Discussion and Future Research

Sensing is a basic measure of the complexity of monitoring and synthesis. In monitoring safety properties, the definition of sensing presented in [2] is not informative, as it gives sensing cost 0 to properties that are satisfied with probability 0. We argue that in the context of monitoring, the definition of sensing cost should consider only computations that satisfy the property, and we study the complexity of computing the sensing cost of a property in the new definition. We distinguish between two approaches to define a probabilistic measure with respect to the set of computations that satisfy a property. We show that while computing the sensing cost according to the new definitions is technically more complicated than in [2], the minimal sensing is still attained by a minimal-size automaton, and it can still be computed in polynomial time.

In synthesis, we introduce a new degree of freedom, namely choosing the outputs when realizing a specification. We study the complexity of finding a minimal-sensing transducer for safety specifications. We show that the minimal-sensing transducer is not necessarily minimal in size. Moreover, interestingly, unlike the case of traditional synthesis, a minimal-sensing transducer need not even correspond to a strategy embodied in the specification deterministic automaton. On the positive side, we show that a minimal-sensing transducer always exists (for a realizable safety specification) and that its size is at most exponential in the size of the minimal-size transducer. We also provide matching lower bounds.

We now turn to discuss some future directions for research.

Non-safety properties. We focus on safety properties. The study in [2] completes the monitoring picture for all other ω-regular properties. We plan to continue the study of synthesis of ω-regular properties. An immediate complication in this setting is that a finite minimal-sensing transducer does not always exists. Indeed, even in the monitoring setting studied in [2], a minimal-sensing automaton does not always exist. Even the decidability of computing the optimal cost remains open.

A trade-off between sensing and quality. Reducing the sensing cost of a transducer can often be achieved by *delaying* the sensing of some letter, thus sensing it less often. This, however, means that eventualities may take longer to be fulfilled, resulting in transducers

of lower quality [1]. We plan to formalize and study the trade-off between the sensing and quality and relate it to the trade-offs between size and sensing, as well as between size and quality.

Acknowledgment. We thank Elie de Panafieu for helpful discussions.

References

1 S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In *20th TACAS*, 2014.
2 S. Almagor, D. Kuperberg, and O. Kupferman. Regular sensing. In *34th FSTTCS*, pages 161–173, 2014.
3 D.L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.
4 P. Flajolet and R. Sedgewick. Analytic combinatorics: functional equations, rational and algebraic functions. 2001.
5 C. Grinstead and J. Laurie Snell. 11:Markov chains. In *Introduction to Probability*. American Mathematical Society, 1997.
6 J. Goubault. Rigid E-Unifiability is DEXPTIME-Complete. In *9th LICS*, pages 498–506, 1994.
7 K. Havelund and G. Rosu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 6(2):18–173, 2004.
8 G. Kindler. *Property Testing, PCP, and Juntas*. PhD thesis, Tel Aviv University, 2002.
9 O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
10 E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.
11 C. Mauduit and A. Sárköz. On finite pseudorandom binary sequences. i. measure of pseudorandomness, the legendre symbol. *Acta Arith.*, 82(4):365–377, 1997.
12 S. Muthukrishnan. Theory of data stream computing: where to go. In *Proc. 30th PODS*, pages 317–319, 2011.
13 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
14 M.F. Roy and A. Szpirglas. Complexity of the computation on real algebraic numbers. *J. Symb. Comput.*, 10(1):39–52, 1990.
15 A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

An ω-Algebra for Real-Time Energy Problems

David Cachera, Uli Fahrenberg, and Axel Legay

Irisa / Inria / École normale supérieure, Rennes, France
{david.cachera,ulrich.fahrenberg,axel.legay}@irisa.fr

Abstract

We develop a *-continuous Kleene ω-algebra of real-time energy functions. Together with corresponding automata, these can be used to model systems which can consume and regain energy (or other types of resources) depending on available time. Using recent results on *-continuous Kleene ω-algebras and computability of certain manipulations on real-time energy functions, it follows that reachability and Büchi acceptance in real-time energy automata can be decided in a static way which only involves manipulations of real-time energy functions.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Energy problem, Real time, Star-continuous Kleene algebra

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.394

1 Introduction

Energy and *resource management* problems are important in areas such as embedded systems or autonomous systems. They are concerned with the following types of questions:

- *Can the system reach a designated state without running out of energy before?*
- *Can the system reach a designated state within a specified time limit without running out of energy?*
- *Can the system repeatedly accomplish certain designated tasks without ever running out of energy?*

Instead of energy, these questions can also be asked using other resources, for example money or fuel.

As an example, imagine a satellite like in Fig. 1 which is being sent up into space. In its initial state when it has arrived at its orbit, its solar panels are still folded, hence no (electrical) energy is generated. Now it needs to unfold its solar panels and rotate itself and its panels into a position orthogonal to the sun's rays (for maximum energy yield). These operations require energy which hence must be provided by a battery, and there may be some operational requirements which state that they have to be completed within a given time limit. To minimize weight, one will generally be interested to use a battery which is as little as possible.

Figure 2 shows a simple toy model of such a satellite's initial operations. We assume that it opens its solar panels in two steps; after the first step they are half open and afterwards fully open, and that it can rotate into orthogonal position at any time. The numbers within the states signify energy gain per time unit, so that for example in the half-open state, the satellite gains 2 energy units per time unit before rotation and 4 after rotation. The (negative) numbers at transitions signify the energy cost for taking that transition, hence it costs 20 energy units to open the solar panels and 10 to rotate.

Now if the satellite battery has sufficient energy, then we can follow any path from the initial to the final state without spending time in intermediate states. A simple inspection reveals that a battery level of 50 energy units is required for this. On the other hand, if

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 394–407

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Figure 1 GPS Block II-F satellite (artist's conception; public domain).

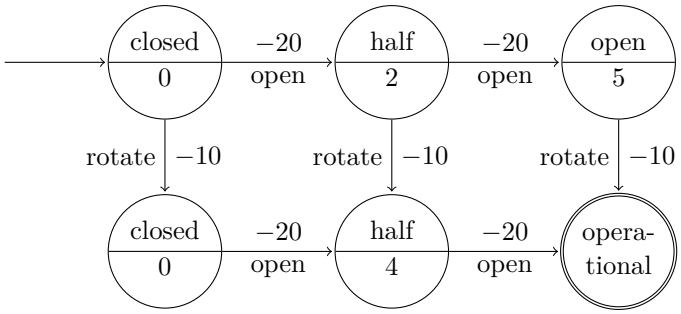


Figure 2 Toy model of the satellite in Fig. 1.

battery level is strictly below 20, then no path is available to the final state. With initial energy level between these two values, the device has to regain energy by spending time in an intermediate state before proceeding to the next one. The optimal path then depends on the available time and the initial energy. For an initial energy level of at least 40, the fastest strategy consists in first opening the panels and then spending 2 time units in state (open|5) to regain enough energy to reach the final state. With the smallest possible battery, storing 20 energy units, 5 time units have to be spent in state (half|2) before passing to (half|4) and spending another 5 time units there.

In this paper we will be concerned with models for such systems which, as in the example, allow to spend time in states to regain energy, some of which has to be spent when taking transitions between states. (Instead of energy, other resource types could be modeled, but we will from now think of it as energy.) We call these models *real-time energy automata*. Their behavior depends, thus, on both the initial energy and the time available; as we have seen in the example, this interplay between time and energy means that even simple models can have rather complicated behaviors. As in the example, we will be concerned with the *reachability* problem for such models, but also with *Büchi acceptance*: whether there exists an infinite run which visits certain designated states infinitely often.

Our methodology is strictly algebraic, using the theory of semiring-weighted automata [8] and extensions developed in [11, 10]. We view the finite behavior of a real-time energy automaton as a function $f(x_0, t)$ which maps initial energy x_0 and available time t to a final energy level, intuitively corresponding to the highest output energy the system can achieve when run with these parameters. We define a composition operator on such *real-time*

energy functions which corresponds to concatenation of real-time energy automata and show that with this composition and maximum as operators, the set of real-time energy functions forms a **-continuous Kleene algebra* [19]. This implies that reachability in real-time energy automata can be decided in a static way which only involves manipulations of real-time energy functions.

To be able to decide Büchi acceptance, we extend the algebraic setting to also encompass real-time energy functions which model infinite behavior. These take as input an initial energy x_0 and time t, as before, but now the output is Boolean: true if these parameters permit an infinite run, false if they do not. We show that both types of real-time energy functions can be organized into a **-continuous Kleene ω-algebra* as defined in [11, 10]. This entails that also Büchi acceptance for real-time energy automata can be decided in a static way which only involves manipulations of real-time energy functions.

The most technically demanding part of the paper is to show that real-time energy functions form a *locally closed semiring* [8, 9]; generalizing some arguments in [9, 10], it then follows that they form a *-continuous Kleene ω-algebra. We conjecture that reachability and Büchi acceptance in real-time energy automata can be decided in exponential time.

Related work. Real-time energy problems have been considered in [20, 5, 4, 6, 15]. These are generally defined on *priced timed automata* [1, 2], a formalism which is more general than ours: it allows for time to be reset and admits several independent time variables (or *clocks*) which can be constrained at transitions. All known decidability results apply to priced timed automata with only *one* time variable; in [6] it is shown that with four time variables, it is undecidable whether there exists an infinite run.

The work which is closest to ours is [4]. Their models are priced timed automata with one time variable and energy updates on transitions, hence a generalization of ours. Using a sequence of complicated ad-hoc reductions, they show that reachability and existence of infinite runs is decidable for their models; whether their techniques apply to general Büchi acceptance is unclear.

Our work is part of a program to make methods from semiring-weighted automata available for energy problems. Starting with [12], we have developed a general theory of *-continuous Kleene ω-algebras [11, 10] and shown that it applies to so-called *energy automata*, which are finite (untimed) automata which allow for rather general *energy transformations* as transition updates. The contribution of this paper is to show that these algebraic techniques can be applied to a real-time setting.

Note that the application of Kleene algebra to real-time and hybrid systems is not a new subject, see for example [17, 7]. However, the work in these papers is based on *trajectories* and *interval predicates*, respectively, whereas our work is on real-time energy *automata*, *i.e.,* at a different level. A more thorough comparison of our work to [17, 7] would be interesting future work.

We acknowledge insightful discussions with Zoltán Ésik on the subject of this paper.

2 Real-Time Energy Automata

Let $\mathbb{R}_{\geq 0} = [0, \infty[$ denote the set of non-negative real numbers, $[0, \infty]$ the set $\mathbb{R}_{\geq 0}$ extended with infinity, and $\mathbb{R}_{\leq 0} =]-\infty, 0]$ the set of non-positive real numbers.

▶ **Definition 1.** A *real-time energy automaton* (RTEA) (S, s_0, F, T, r) consists of a finite set S of *states*, with *initial state* $s_0 \in S$, a subset $F \subseteq S$ of *accepting* states, a finite set $T \subseteq S \times \mathbb{R}_{\leq 0} \times \mathbb{R}_{\geq 0} \times S$ of *transitions*, and a mapping $r : S \to \mathbb{R}_{\geq 0}$ assigning *rates* to states.

A transition (s, p, b, s') is written $s \xrightarrow[b]{p} s'$, p is called its *price* and b its *bound*. We assume $b \geq -p$ for all transitions $s \xrightarrow[b]{p} s'$.

An RTEA is *computable* if all its rates, prices and bounds are computable real numbers.

A *configuration* of an RTEA $A = (S, s_0, F, T, r)$ is an element $(s, x, t) \in C = S \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$. Let $\rightsquigarrow \subseteq C \times C$ be the relation given by $(s, x, t) \rightsquigarrow (s', x', t')$ iff $t' \leq t$ and there is a transition $s \xrightarrow[b]{p} s'$ such that $x + (t - t')r(s) \geq b$ and $x' = x + (t - t')r(s) + p$. Hence $t - t'$ time units are spent in state s and afterwards a transition $s \xrightarrow[b]{p} s'$ is taken.

A *run* in A is a path in the infinite directed graph $(C, \rightsquigarrow)$, *i.e.,* a finite or infinite sequence $(s_1, x_1, t_1) \rightsquigarrow (s_2, x_2, t_2) \rightsquigarrow \cdots$. We are ready to state the decision problems for RTEAs with which we will be concerned. Let $A = (S, s_0, F, T, r)$ be a computable RTEA and $x_0, t, y \in [0, \infty]$ computable numbers.

▶ Problem 1 (State reachability). Does there exist a finite run $(s_0, x_0, t) \rightsquigarrow \cdots \rightsquigarrow (s, x, t')$ in A with $s \in F$?

▶ Problem 2 (Coverability). Does there exist a finite run $(s_0, x_0, t) \rightsquigarrow \cdots \rightsquigarrow (s, x, t')$ in A with $s \in F$ and $x \geq y$?

▶ Problem 3 (Büchi acceptance). Does there exist $s \in F$ and an infinite run $(s_0, x_0, t) \rightsquigarrow (s_1, x_1, t_1) \rightsquigarrow \cdots$ in A in which $s_n = s$ for infinitely many $n \geq 0$?

Note that the coverability problem only asks for the final energy level x to be *above* y; as we are interested in *maximizing* energy, this is natural. Also, state reachability can be reduced to coverability by setting $y = 0$. As the Büchi acceptance problem asks for infinite runs, there is no notion of output energy for this problem.

Asking the Büchi acceptance question for a *finite* available time $t < \infty$ amounts to finding (accepting) *Zeno runs* in the given RTEA, *i.e.,* runs which make infinitely many transitions in finite time. Hence one will usually be interested in Büchi acceptance only for an infinite time horizon. On the other hand, for $t = \infty$, a positive answer to the state reachability problem 1 consists of a finite run $(s_0, t_0, \infty) \rightsquigarrow \cdots \rightsquigarrow (s, x, \infty)$, and as one can delay indefinitely in the state $s \in F$, this leads to an infinite run. Per our definition of $\rightsquigarrow$, such an infinite run will *not* be a positive answer to the Büchi acceptance problem.

3 Weighted Automata over *-Continuous Kleene ω-Algebras

We now turn our attention to the algebraic setting of *-continuous Kleene algebras and related structures and review some results on *-continuous Kleene algebras and *-continuous Kleene ω-algebras which we will need in the sequel.

3.1 *-Continuous Kleene Algebras

An *idempotent semiring* [16] $S = (S, \vee, \cdot, \bot, 1)$ consists of an idempotent commutative monoid $(S, \vee, \bot)$ and a monoid $(S, \cdot, 1)$ such that the distributive and zero laws

$$x(y \vee z) = xy \vee xz \qquad (y \vee z)x = yx \vee zx \qquad \bot x = \bot = x\bot$$

hold for all $x, y, z \in S$. It follows that the product operation distributes over all finite suprema. Each idempotent semiring S is partially ordered by the relation $x \leq y$ iff $x \vee y = y$, and then sum and product preserve the partial order and $\bot$ is the least element.

A *Kleene algebra* [19] is an idempotent semiring $S = (S, \vee, \cdot, \bot, 1)$ equipped with an operation $^* : S \to S$ such that for all $x, y \in S$, yx^* is the least solution of the fixed point

equation $z = zx \vee y$ and x^*y is the least solution of the fixed point equation $z = xz \vee y$ with respect to the order $\leq$.

A **-continuous Kleene algebra* [19] is a Kleene algebra $S = (S, \vee, \cdot, ^*, \bot, 1)$ in which the infinite suprema $\bigvee\{x^n \mid n \geq 0\}$ exist for all $x \in S$, $x^* = \bigvee\{x^n \mid n \geq 0\}$ for every $x \in S$, and product preserves such suprema: for all $x, y \in S$,

$$y(\bigvee_{n\geq 0} x^n) = \bigvee_{n\geq 0} yx^n \quad \text{and} \quad (\bigvee_{n\geq 0} x^n)y = \bigvee_{n\geq 0} x^n y\,.$$

An idempotent semiring $S = (S, \vee, \cdot, \bot, 1)$ is said to be *locally closed* [9] if it holds that for every $x \in S$, there exists $N \geq 0$ so that $\bigvee_{n=0}^{N} x^n = \bigvee_{n=0}^{N+1} x^n$. In any locally closed idempotent semiring, we may define a *-operation by $x^* = \bigvee_{n\geq 0} x^n$.

▶ **Lemma 2.** *Any locally closed idempotent semiring is a *-continuous Kleene algebra.*

3.2 *-Continuous Kleene ω-Algebras

An *idempotent semiring-semimodule pair* [14, 3] (S, V) consists of an idempotent semiring $S = (S, \vee, \cdot, \bot, 1)$ and a commutative idempotent monoid $V = (V, \vee, \bot)$ which is equipped with a left S-action $S \times V \to V$, $(s, v) \mapsto sv$, satisfying

$$\begin{array}{lll} (s \vee s')v = sv \vee s'v & s(v \vee v') = sv \vee sv' & \bot s = \bot \\ (ss')v = s(s'v) & s\bot = \bot & 1v = v \end{array}$$

for all $s, s' \in S$ and $v \in V$. In that case, we also call V a *(left) S-semimodule*.

A *generalized *-continuous Kleene algebra* [11] is an idempotent semiring-semimodule pair (S, V) where $S = (S, \vee, \cdot, ^*, \bot, 1)$ is a *-continuous Kleene algebra such that for all $x, y \in S$ and for all $v \in V$,

$$xy^*v = \bigvee_{n\geq 0} xy^n v$$

A **-continuous Kleene ω-algebra* [11] consists of a generalized *-continuous Kleene algebra (S, V) together with an infinite product operation $S^\omega \to V$ which maps every infinite sequence $x_0, x_1, \ldots$ in S to an element $\prod_{n\geq 0} x_n$ of V. The infinite product is subject to the following conditions (see [11] for motivation):

- For all $x_0, x_1, \ldots \in S$, $\prod_{n\geq 0} x_n = x_0 \prod_{n\geq 0} x_{n+1}$. (C1)
- Let $x_0, x_1, \ldots \in S$ and $0 = n_0 \leq n_1 \leq \cdots$ a sequence which increases without a bound. Let $y_k = x_{n_k} \cdots x_{n_{k+1}-1}$ for all $k \geq 0$. Then $\prod_{n\geq 0} x_n = \prod_{k\geq 0} y_k$. (C2)
- For all $x_0, x_1, \ldots, y, z \in S$, $\prod_{n\geq 0}(x_n(y \vee z)) = \bigvee_{x'_0, x'_1, \ldots \in \{y,z\}} \prod_{n\geq 0} x_n x'_n$. (C3)
- For all $x, y_0, y_1, \ldots \in S$, $\prod_{n\geq 0} x^* y_n = \bigvee_{k_0, k_1, \ldots \geq 0} \prod_{n\geq 0} x^{k_n} y_n$. (C4)

3.3 Matrix Semiring-Semimodule Pairs

For any idempotent semiring S and $n \geq 1$, we can form the matrix semiring $S^{n\times n}$ whose elements are $n \times n$-matrices of elements of S and whose sum and product are given as the

usual matrix sum and product. It is known [18] that when S is a *-continuous Kleene algebra, then $S^{n\times n}$ is also a *-continuous Kleene algebra, with the *-operation defined by

$$M^*_{i,j} = \bigvee_{m\geq 0} \bigvee \left\{ M_{k_1,k_2} M_{k_2,k_3} \cdots M_{k_{m-1},k_m} \;\middle|\; 1 \leq k_1, \ldots, k_m \leq n, k_1 = i, k_m = j \right\} \tag{1}$$

for all $M \in S^{n\times n}$ and $1 \leq i, j \leq n$. Also, if $n \geq 2$ and $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, where a and d are square matrices of dimension less than n, then

$$M^* = \begin{pmatrix} (a \vee bd^*c)^* & (a \vee bd^*c)^*bd^* \\ (d \vee ca^*b)^*ca^* & (d \vee ca^*b)^* \end{pmatrix} \tag{2}$$

For any idempotent semiring-semimodule pair (S, V) and $n \geq 1$, we can form the matrix semiring-semimodule pair $(S^{n\times n}, V^n)$ whose elements are $n \times n$-matrices of elements of S and n-dimensional (column) vectors of elements of V, with the action of $S^{n\times n}$ on V^n given by the usual matrix-vector product.

When (S, V) is a *-continuous Kleene ω-algebra, then $(S^{n\times n}, V^n)$ is a generalized *-continuous Kleene algebra [11]. By [11, Lemma 17], there is an ω-operation on $S^{n\times n}$ defined by

$$M^\omega_i = \bigvee_{1\leq k_1,k_2,\ldots\leq n} M_{i,k_1} M_{k_1,k_2} \cdots$$

for all $M \in S^{n\times n}$ and $1 \leq i \leq n$. Also, if $n \geq 2$ and $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, where a and d are square matrices of dimension less than n, then

$$M^\omega = \begin{pmatrix} (a \vee bd^*c)^\omega \vee (a \vee bd^*c)^*bd^\omega \\ (d \vee ca^*b)^\omega \vee (d \vee ca^*b)^*ca^\omega \end{pmatrix} \tag{3}$$

It can be shown [13] that the number of semiring computations required in the computation of M^* and M^ω in (2) and (3) is $O(n^3)$ and $O(n^4)$, respectively.

3.4 Weighted automata

Let (S, V) be a *-continuous Kleene ω-algebra and $A \subseteq S$ a subset. We write $\langle A\rangle$ for the set of all finite suprema $a_1 \vee \cdots \vee a_m$ with $a_i \in A$ for each $i = 1, \ldots, m$.

A *weighted automaton* [8] over A of dimension $n \geq 1$ is a tuple (α, M, k), where $\alpha \in \{\bot, 1\}^n$ is the initial vector, $M \in \langle A\rangle^{n\times n}$ is the transition matrix, and k is an integer $0 \leq k \leq n$. Combinatorially, this may be represented as a transition system whose set of states is $\{1, \ldots, n\}$. For any pair of states i, j, the transitions from i to j are determined by the entry $M_{i,j}$ of the transition matrix: if $M_{i,j} = a_1 \vee \cdots \vee a_m$, then there are m transitions from i to j, respectively labeled $a_1, \ldots, a_n$. The states i with $\alpha_i = 1$ are *initial*, and the states $\{1, \ldots, k\}$ are *accepting*.

The *finite behavior* of a weighted automaton $A = (\alpha, M, k)$ is defined to be

$$|A| = \alpha M^* \kappa\,,$$

where $\kappa \in \{\bot, 1\}^n$ is the vector given by $\kappa_i = 1$ for $i \leq k$ and $\kappa_i = \bot$ for $i > k$. (Note that α has to be used as a *row* vector for this multiplication to make sense.) It is clear by (1) that $|A|$ is the supremum of the products of the transition labels along all paths in A from any initial to any accepting state.

The *Büchi behavior* of a weighted automaton $A = (\alpha, M, k)$ is defined to be

$$\|A\| = \alpha \begin{pmatrix} (a + bd^*c)^\omega \\ d^*c(a + bd^*c)^\omega \end{pmatrix},$$

where $a \in \langle A \rangle^{k\times k}$, $b \in \langle A \rangle^{k\times(n-k)}$, $c \in \langle A \rangle^{(n-k)\times n}$ and $d \in \langle A \rangle^{(n-k)\times(n-k)}$ are such that $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$. Note that M is split in submatrices $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ precisely so that a contains transitions between accepting states and d contains transitions between non-accepting states. By [11, Thm. 20], $\|A\|$ is the supremum of the products of the transition labels along all infinite paths in A from any initial state which infinitely often visit an accepting state.

4 Real-Time Energy Functions

Let $L = [0, \infty]_\perp$ denote the set of non-negative real numbers extended with a bottom element $\perp$ and a top element ∞. We use the standard order on L, *i.e.,* the one on $\mathbb{R}_{\geq 0}$ extended by declaring $\perp \leq x \leq \infty$ for all $x \in L$. L is a complete lattice, whose suprema we will denote by $\vee$ for binary and $\bigvee$ for general supremum. For convenience we also extend the addition on $\mathbb{R}_{\geq 0}$ to L by declaring that $\perp + x = x + \perp = \perp$ for all $x \in L$ and $\infty + x = x + \infty = \infty$ for all $x \in L \setminus \{\perp\}$. Note that $\perp + \infty = \infty + \perp = \perp$.

Let $\mathcal{F}$ denote the set of monotonic functions $f : L \times [0, \infty] \to L$ (with the product order on $L \times [0, \infty]$) for which $f(\perp, t) = \perp$ for all $t \in L$. We will frequently write such functions in curried form, using the equivalence $\langle L \times [0, \infty] \to L \rangle \approx \langle [0, \infty] \to L \to L \rangle$.

4.1 Linear Real-Time Energy Functions

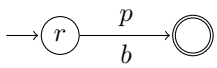
We will be considered with the subset of functions in $\mathcal{F}$ consisting of *real-time energy functions* (RTEFs). These correspond to functions expressed by RTEAs, and we will construct them inductively. We start with atomic RTEFs:

▶ **Definition 3.** Let $r, b, p \in \mathbb{R}$ with $r \geq 0$, $p \leq 0$ and $b \geq -p$. An *atomic real-time energy function* is an element f of $\mathcal{F}$ such that $f(\perp, t) = \perp$, $f(\infty, t) = \infty$, $f(x, \infty) = \infty$, and

$$f(x,t) = \begin{cases} x + rt + p & \text{if } x + rt \geq b\,, \\ \perp & \text{otherwise} \end{cases}$$

for all $x, t \in \mathbb{R}_{\geq 0}$. The numbers r, b and p are respectively called the *rate*, *bound* and *price* of f. We denote by $\mathcal{A} \subseteq \mathcal{F}$ the set of atomic real-time energy functions.

These functions arise from RTEAs with one transition:



Non-negativity of r ensures that atomic RTEFs are monotonic. In our examples, when the bound is not explicitly mentioned it corresponds to the lowest possible one: $b = -p$.

Atomic RTEFs are naturally combined along acyclic paths by means of a composition operator. Intuitively, a composition of two successive atomic RTEFs determines the optimal output energy one can get after spending some time in either one or both locations of the corresponding automaton. This notion of composition is naturally extended to all functions in $\mathcal{F}$, and formally defined as follows.

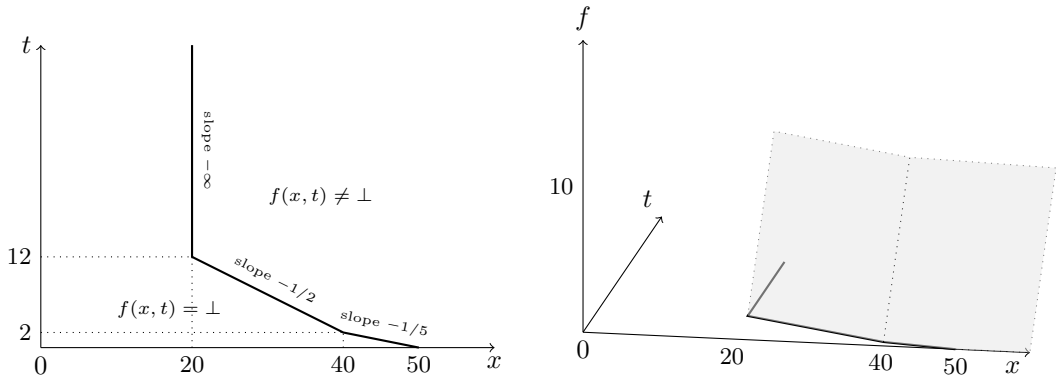


Figure 3 Graphical representation of the linear RTEF from Example 7.

▶ **Definition 4.** The *composition* of $f, g \in \mathcal{F}$ is the element $f \triangleright g$ of $\mathcal{F}$ such that

$$\forall t \in [0,\infty] : f \triangleright g(t) = \bigvee_{t_1+t_2=t} g(t_2) \circ f(t_1) \tag{4}$$

Note that composition is written in diagrammatic order. Uncurrying the equation, we see that $f \triangleright g(x,t) = \bigvee_{t_1+t_2=t} g(f(x,t_1),t_2)$.

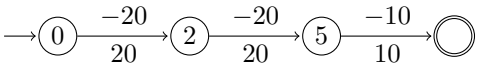
Let $\mathbf{1}, \bot \in \mathcal{F}$ be the functions defined by $\mathbf{1}(t)(x) = x$ and $\bot(t)(x) = \bot$ for all x, t.

▶ **Lemma 5.** *The $\triangleright$ operator is associative, with $\mathbf{1}$ as neutral and $\bot$ as absorbing elements.*

Compositions of atomic RTEFs along paths are called *linear* RTEFs:

▶ **Definition 6.** A *linear real-time energy function* is a finite composition $f_1 \triangleright f_2 \triangleright \cdots \triangleright f_n$ of atomic RTEFs.

▶ **Example 7.** As an example, and also to show that linear RTEFs can have quite complex behavior, we show the linear RTEF associated to one of the paths in the satellite example of the introduction. Consider the following (linear) RTEA:



Its linear RTEF f can be computed as follows:

$$f(x,t) = \begin{cases} \bot & \text{if } x < 20 \text{ or } (20 \le x < 40 \text{ and } x + 2t < 44) \\ & \quad \text{or } (x \ge 40 \text{ and } x + 5t < 50) \\ 2.5x + 5t - 110 & \text{if } 20 \le x < 40 \text{ and } x + 2t \ge 44 \\ x + 5t - 50 & \text{if } x \ge 40 \text{ and } x + 5t \ge 50 \end{cases}$$

We show a graphical representation of f on Fig. 3. The left part of the figure shows the boundary between two regions in the (x,t) plane, corresponding to the minimal value 0 achieved by the function. Below this boundary, no path exists through the corresponding RTEA. Above, the function is linear in x and t. The coefficient of t corresponds to the maximal rate in the RTEA; the coefficient of x depends on the relative position of x with respect to the bounds b_i.

4.2 Normal Form

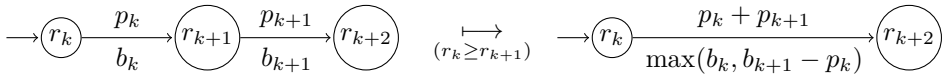
Next we need to see that all linear RTEFs can be converted to a *normal form*:

▶ **Definition 8.** A sequence $f_1, \ldots, f_n$ of atomic RTEFs, with rates, bounds and prices $r_1, \ldots, r_n$, $b_1, \ldots, b_n$ and $p_1, \ldots, p_n$, respectively, is in *normal form* if

- $r_1 < \cdots < r_n$,
- $b_1 \leq \cdots \leq b_n$, and
- $p_1 = \cdots = p_{n-1} = 0$.

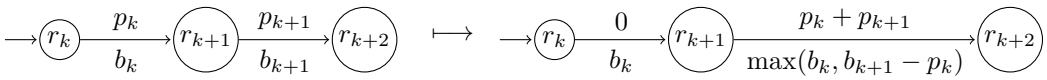
▶ **Lemma 9.** *For any linear RTEF f there exists a sequence $f_1, \ldots, f_n$ of atomic RTEFs in normal form such that $f = f_1 \triangleright \cdots \triangleright f_n$.*

Proof sketch. Let $f = f_1 \triangleright \cdots \triangleright f_n$, where $f_1, \ldots, f_n$ are atomic RTEFs and assume $f_1, \ldots, f_n$ is not in normal form. If there is an index $k \in \{1, \ldots, n-1\}$ with $r_k \geq r_{k+1}$, then we can use the following transformation to remove the state with rate r_{k+1}:



Informally, any run through the RTEA for $f_1 \triangleright \cdots \triangleright f_n$ which maximizes output energy will spend no time in the state with rate r_{i+1}, as this time may as well be spent in the state with rate r_i without lowering output energy.

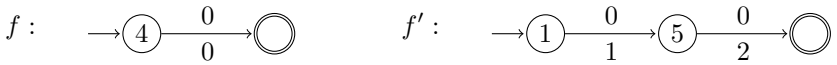
To ensure the last two conditions of Definition 8, we use the following transformation:



Any run through the original RTEA can be copied to the other and vice versa, hence also this transformation does not change the values of f. ◀

▶ **Definition 10.** Let $f_1, \ldots, f_n$ and $f'_1, \ldots, f'_{n'}$ be normal-form sequences of atomic RTEFs with rate sequences $r_1 < \cdots < r_n$ and $r'_1 < \cdots < r'_{n'}$, respectively. Then $f_1, \ldots, f_n$ is *not better than* $f'_1, \ldots, f'_{n'}$, denoted $(f_1, \ldots, f_n) \preceq (f'_1, \ldots, f'_{n'})$, if $r_n \leq r'_{n'}$.

Note that $(f_1, \ldots, f_n) \preceq (f'_1, \ldots, f'_{n'})$ does not imply $f_1 \triangleright \cdots \triangleright f_n \leq f'_1 \triangleright \cdots \triangleright f'_{n'}$ even for very simple functions. For a counterexample, consider the two following linear RTEFs $f = f_1$, $f' = f'_1 \triangleright f'_2$ with corresponding RTEAs:

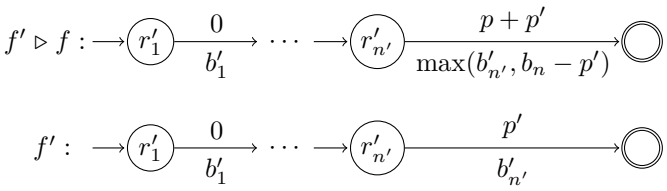


We have $(f_1) \preceq (f'_1, f'_2)$, and for $x \geq 2$, $f(x,t) = x + 4t$ and $f'(x,t) = x + 5t$, hence $f(x,t) \leq f'(x,t)$. But $f(0,1) = 4$, whereas $f'(0,1) = \bot$.

▶ **Lemma 11.** *If $f = f_1 \triangleright \cdots \triangleright f_n$ and $f' = f'_1 \triangleright \cdots \triangleright f'_{n'}$ are such that $(f_1, \ldots, f_n) \preceq (f'_1, \ldots, f'_{n'})$, then $f' \triangleright f \leq f'$.*

Proof. Let $r_1 < \cdots < r_n$ and $r'_1 < \cdots < r'_{n'}$ be the corresponding rate sequences, then $r_n \leq r'_{n'}$. The RTEAs for $f' \triangleright f$ and f' are as follows, where we have transformed the former

to normal form using that for all indices i, $r_i \le r_n \le r'_{n'}$:



As $p+p' \le p'$ (because $p \le 0$) and $\max(b'_{n'}, b_n - p') \ge b'_{n'}$, it is clear that $f' \triangleright f(x,t) \le f'(x,t)$ for all $x \in L$, $t \in [0,\infty]$. ◀

4.3 General Real-Time Energy Functions

We now consider all paths that may arise in a real-time energy automaton. When two locations of an automaton may be joined by two distinct paths, the optimal output energy is naturally obtained by taking the maximum over both paths. This gives rise to the following definition.

▶ **Definition 12.** Let $f, g \in \mathcal{F}$. The function $f \vee g$ is defined as the pointwise supremum:

$$\forall t \in [0,\infty] : (f \vee g)(t) = f(t) \vee g(t)$$

▶ **Lemma 13.** *With operations $\vee$ and $\triangleright$, $\mathcal{F}$ forms a complete lattice and an idempotent semiring, with $\bot$ as unit for $\vee$ and $\mathbf{1}$ as unit for $\triangleright$.*

Finally, a cycle in an RTEA results in a *-operation:

▶ **Definition 14.** Let $f \in \mathcal{F}$. The Kleene star of f is the function $f^* \in \mathcal{F}$ such that

$$\forall t \in [0,\infty] : f^*(t) = \bigvee_{n \ge 0} f^n(t)$$

Note that f^* is defined for all $f \in \mathcal{F}$ because $\mathcal{F}$ is a complete lattice. We can now define the set of general real-time energy functions, corresponding to general RTEAs:

▶ **Definition 15.** The set $\mathcal{E}$ of *real-time energy functions* is the subsemiring of $\mathcal{F}$ generated by $\mathcal{A}$, *i.e.,* the subset of $\mathcal{F}$ inductively defined by

- $\mathcal{A} \subseteq \mathcal{E}$,
- if $f, g \in \mathcal{E}$, then $f \triangleright g \in \mathcal{E}$ and $f \vee g \in \mathcal{E}$.

We will show below that $\mathcal{E}$ is locally closed, which entails that for each $f \in \mathcal{E}$, also $f^* \in \mathcal{E}$, hence $\mathcal{E}$ indeed encompasses all RTEFs.

▶ **Lemma 16.** *For every $f \in \mathcal{E}$ there exists $N \ge 0$ so that $f^* = \bigvee_{n=0}^{N} f^n$.*

Proof. By distributivity, we can write f as a finite supremum $f = \bigvee_{k=1}^{m} f_k$ of linear energy functions $f_1, \dots, f_m$. For each $k = 1, \dots, m$, let $f_k = f_{k,1} \triangleright \cdots \triangleright f_{k,n_k}$ be a normal-form representation. By re-ordering the f_k if necessary, we can assume that $(f_{k,1}, \dots, f_{k,n_k}) \preceq (f_{k+1,1}, \dots, f_{k+1,n_{k+1}})$ for every $k = 1, \dots, n-1$.

We first show that $f^* \le \bigvee_{0 \le n_1, \dots, n_m \le 1} f_1^{n_1} \triangleright \cdots \triangleright f_m^{n_m}$: The expansion of $f^* = (\bigvee_{k=1}^{m} f_k)^*$ is an infinite supremum of finite compositions $f_{i_1} \triangleright \cdots \triangleright f_{i_p}$. By Lemma 11, any occurrence of $f_{i_j} \triangleright f_{i_{j+1}}$ in such compositions with $i_j \ge i_{j+1}$ can be replaced by $f_{i_{j+1}}$. The compositions which are left have $i_j < i_{j+1}$ for every j, so the claim follows.

Now $\bigvee_{0 \le n_1, \dots, n_m \le 1} f_1^{n_1} \triangleright \cdots \triangleright f_m^{n_m} \le \bigvee_{n=0}^{m} \left(\bigvee_{k=1}^{m} f_k\right)^n = \bigvee_{n=0}^{m} f^n \le f^*$, so with $N = m$ the proof is complete. ◀

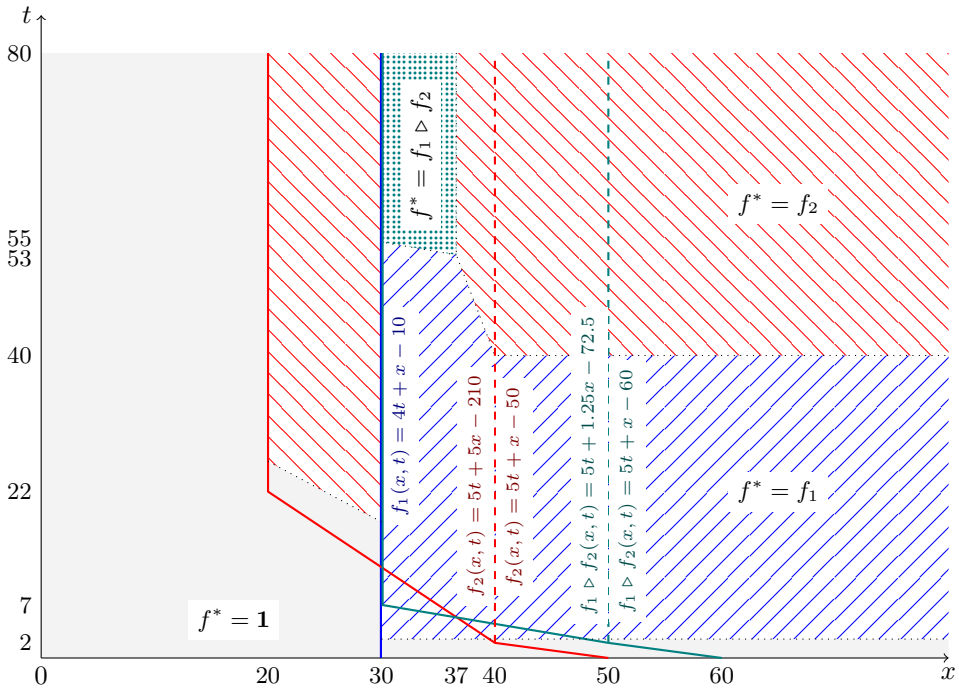
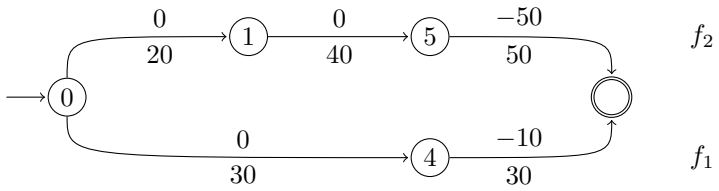


Figure 4 Computation of f^* from Example 18.

▶ **Corollary 17.** *$\mathcal{E}$ is locally closed, hence a *-continuous Kleene algebra.*

Proof. For every $f \in \mathcal{E}$ there is $N \geq 0$ so that $f^* = \bigvee_{n=0}^{N} f^n$ (Lemma 16), hence $\bigvee_{n=0}^{N} f^n = \bigvee_{n=0}^{N+1} f^n$. Thus $\mathcal{E}$ is locally closed, and by Lemma 2, a *-continuous Kleene algebra. ◀

▶ **Example 18.** To illustrate, we compute the Kleene star of the supremum $f = f_1 \vee f_2$ of two linear RTEFs as below. These are slight modifications of some RTEFs from the satellite example, modified to make the example more interesting:



These functions are in normal form and $f_1 \preceq f_2$. Lemma 16 and its proof allow us to conclude that $f^* = \mathbf{1} \vee f_1 \vee f_2 \vee f_1 \triangleright f_2$. Figure 4 shows the boundaries of definition of these functions and the regions in the (x,t) plane where each of them dominates.

4.4 Infinite Products

Let $\mathbb{B} = \{\mathbf{ff}, \mathbf{tt}\}$ denote the Boolean lattice with standard order $\mathbf{ff} < \mathbf{tt}$. Let $\mathcal{V}$ denote the set of monotonic functions $v : L \times [0,\infty] \to \mathbb{B}$ for which $v(\bot, t) = \bot$ for all $t \in L$. We define an infinite product operation $\mathcal{F}^\omega \to \mathcal{V}$:

▶ **Definition 19.** For an infinite sequence of functions $f_0, f_1, \ldots \in \mathcal{F}$, $\prod_{n\geq 0} f_n \in \mathcal{V}$ is the function defined for $x \in L$, $t \in [0,\infty]$ by $\prod_{n\geq 0} f_n(x,t) = \mathbf{tt}$ iff there is an infinite sequence $t_0, t_1, \ldots \in [0,\infty]$ such that $\sum_{n=0}^{\infty} t_n = t$ and for all $n \geq 0$, $f_n(t_n) \circ \cdots \circ f_0(t_0)(x) \neq \bot$.

Hence $\prod_{n\geq 0} f_n(x,t) = \mathbf{tt}$ iff in the infinite composition $f_0 \triangleright f_1 \triangleright \cdots (x,t)$, all finite prefixes have values $\neq \bot$. There is a (left) action of $\mathcal{F}$ on $\mathcal{V}$ given by $(f,v) \mapsto f \triangleright v$, where the composition $f \triangleright v$ is given by the same formula as composition $\triangleright$ on $\mathcal{F}$. Let $\bot \in \mathcal{V}$ denote the function given by $\bot(x,t) = \mathbf{ff}$.

▶ **Lemma 20.** *With the $\mathcal{F}$-action $\triangleright$, $\vee$ as addition, and $\bot$ as unit, $\mathcal{V}$ is an idempotent left $\mathcal{F}$-semimodule.*

Let $\mathcal{U} \subseteq \mathcal{V}$ be the $\mathcal{F}$-subsemimodule generated by $\mathcal{E} \subseteq \mathcal{F}$. Then $\mathcal{U}$ is an idempotent left $\mathcal{E}$-semimodule.

▶ **Proposition 21.** *$(\mathcal{E},\mathcal{U})$ forms a *-continuous Kleene ω-algebra.*

5 Decidability

We can now apply the results of Section 3.4 to see that our decision problems as stated at the end of Section 2 are decidable. Let Let $A = (S, s_0, F, T, r)$ be an RTEA, with matrix representation (α, M, K), and $x_0, t, y \in [0,\infty]$.

▶ **Theorem 22.** *There exists a finite run $(s_0,x_0,t) \rightsquigarrow \cdots \rightsquigarrow (s,x,t')$ in A with $s \in F$ iff $|A|(x_0,t) > \bot$.*

▶ **Theorem 23.** *There exists a finite run $(s_0,x_0,t) \rightsquigarrow \cdots \rightsquigarrow (s,x,t')$ in A with $s \in F$ and $x \geq y$ iff $|A|(x_0,t) \geq y$.*

▶ **Theorem 24.** *There exists $s \in F$ and an infinite run $(s_0,x_0,t) \rightsquigarrow (s_1,x_1,t_1) \rightsquigarrow \cdots$ in A in which $s_n = s$ for infinitely many $n \geq 0$ iff $\|A\|(x_0,t) = \top$.*

▶ **Theorem 25.** *Problems 1, 2 and 3 from Section 2 are decidable.*

Proof sketch. We have seen in the examples that RTEFs are *piecewise linear*, *i.e.,* composed of a finite number of (affine) linear functions which are defined on polygonal regions in the (x,t)-plane. Such functions can be represented using the (finitely many) corner points of these regions together with their values at these corner points. (In case some regions are not convex or disconnected, they have to be split into convex regions.)

It is clear that computable atomic RTEFs are computable piecewise linear (*i.e.,* all numbers in their finite representation are computable), and that compositions and suprema of computable piecewise linear are again computable piecewise linear. Using Lemma 16, we see that all functions in M^* are computable piecewise linear. ◀

6 Conclusion

We have developed an algebraic methodology for deciding reachability and Büchi problems on a class of weighted real-time models where the weights represent energy or similar quantities. The semantics of such systems is modeled by real-time energy functions which map initial energy of the system and available time to the maximal final energy level. We have shown that these real-time energy functions form a *-continuous Kleene ω-algebra, which entails that reachability and Büchi acceptance can be decided in a static way which only involves manipulations of energy functions.

We have seen that the necessary manipulations of real-time energy functions are computable, and in fact we conjecture that our method leads to an exponential-time algorithm for deciding reachability and Büchi acceptance in real-time energy automata. This is due to

the fact that operations on real-time energy functions can be done in time linear in the size of their representation, and the representation size of compositions and suprema of real-time energy functions is a linear function of the representation size of the operands. In future work, we plan to do a careful complexity analysis which could confirm this result and to implement our algorithms to see how it fares in practice.

This paper constitutes the first application of methods from Kleene algebra to a timed-automata like formalism. In future work, we plan to lift some of the restrictions of the current model and extend it to allow for time constraints and resets à la timed automata. We also plan to extend this work with action labels, which algebraically means passing from the semiring of real-time energy functions to the one of formal power series over these functions. In applications, this means that instead of asking for existence of accepting runs, one is asking for controllability.

References

1 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *HSCC*, volume 2034 of *LNCS*, pages 49–62. Springer, 2001.

2 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

3 Stephen L. Bloom and Zoltán Ésik. *Iteration Theories*. Springer, 1993.

4 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In *HSCC*, pages 61–70. ACM, 2010.

5 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *FORMATS*, volume 5215 of *LNCS*, pages 33–47. Springer, 2008.

6 Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Lower-bound-constrained runs in weighted timed automata. *Perform. Eval.*, 73:91–109, 2014.

7 Brijesh Dongol, Ian J. Hayes, Larissa Meinicke, and Kim Solin. Towards an algebra for real-time programs. In *RAMiCS*, volume 7560 of *LNCS*, pages 50–65. Springer, 2012.

8 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Springer, 2009.

9 Zoltá Ésik and Werner Kuich. Locally closed semirings. *Monatsh. Math.*, 137(1):21–29, 2002.

10 Zoltán Ésik, Uli Fahrenberg, and Axel Legay. *-continuous Kleene ω-algebras. *CoRR*, abs/1501.01118, 2015. `http://arxiv.org/abs/1501.01118`.

11 Zoltán Ésik, Uli Fahrenberg, and Axel Legay. Star-continuous Kleene omega-algebras. In *DLT*, volume 9168 of *LNCS*, pages 240–251. Springer, 2015.

12 Zoltán Ésik, Uli Fahrenberg, Axel Legay, and Karin Quaas. Kleene algebras and semimodules for energy problems. In *ATVA*, volume 8172 of *LNCS*, pages 102–117. Springer, 2013.

13 Zoltán Ésik and Werner Kuich. *Modern Automata Theory*. TU Wien, 2007. `http://dmg.tuwien.ac.at/kuich/mat.pdf`.

14 Zoltán Ésik and Werner Kuich. On iteration semiring-semimodule pairs. *Semigroup Forum*, 75:129–159, 2007.

15 Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jiří Srba. Energy games in multiweighted automata. In *ICTAC*, volume 6916 of *LNCS*, pages 95–115. Springer, 2011.

16 Jonathan S. Golan. *Semirings and their Applications*. Springer, 1999.

17 Peter Höfner and Bernhard Möller. An algebra of hybrid systems. *J. Log. Alg. Prog.*, 78(2):74–97, 2009.

18 Dexter Kozen. On Kleene algebras and closed semirings. In *MFCS*, pages 26–47, 1990.

19 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.

20 Karin Quaas. On the interval-bound problem for weighted timed automata. In *LATA*, volume 6638 of *LNCS*, pages 452–464. Springer, 2011.

Parameterized Complexity of Secluded Connectivity Problems*

Fedor V. Fomin[1,2], Petr A. Golovach[1,2], Nikolay Karpov[2], and Alexander S. Kulikov[2]

1 **Department of Informatics, University of Bergen, Norway,** {fedor.fomin,petr.golovach}@uib.no

2 **Steklov Institute of Mathematics at St.Petersburg, Russian Academy of Sciences, Russia,** {kimaska,alexander.s.kulikov}@gmail.com

Abstract

The Secluded Path problem introduced by Chechik et al. in [ESA 2013] models a situation where a sensitive information has to be transmitted between a pair of nodes along a path in a network. The measure of the quality of a selected path is its *exposure*, which is the total weight of vertices in its closed neighborhood. In order to minimize the risk of intercepting the information, we are interested in selecting a *secluded* path, i.e. a path with a small exposure. Similarly, the Secluded Steiner Tree problem is to find a tree in a graph connecting a given set of terminals such that the exposure of the tree is minimized. In this work, we obtain the following results about parameterized complexity of secluded connectivity problems.

We start from an observation that being parameterized by the size of the exposure, the problem is fixed-parameter tractable (FPT). More precisely, we give an algorithm deciding if a graph G with a given cost function $\omega\colon V(G)\to\mathbb{N}$ contains a secluded path of exposure at most k with the cost at most C in time $\mathcal{O}(3^{k/3}\cdot(n+m)\log W)$, where W is the maximum value of ω on an input graph G. Similarly, Secluded Steiner Tree is solvable in time $\mathcal{O}(2^k k^2\cdot(n+m)\log W)$.

The main result of this paper is about "above guarantee" parameterizations for secluded problems. We show that Secluded Steiner Tree is FPT being parameterized by $r+p$, where p is the number of the terminals, ℓ the size of an optimum Steiner tree, and $r=k-\ell$. We complement this result by showing that the problem is co-W[1]-hard when parameterized by r only.

We also investigate Secluded Steiner Tree from kernelization perspective and provide several lower and upper bounds when parameters are the treewidth, the size of a vertex cover, maximum vertex degree and the solution size. Finally, we refine the algorithmic result of Chechik et al. by improving the exponential dependence from the treewidth of the input graph.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Secluded path, Secluded Steiner tree, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.408

1 Introduction

Secluded Path and Secluded Steiner Tree problems were introduced in Chechik et al. in [8]. In the Secluded Path problem, for given vertices s and t of a graph G, the task is to

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959 and the Government of the Russian Federation (grant 14.Z50.31.0030).

find an s,t-path with the minimum *exposure*, i.e. a path P such that the number of vertices from P plus the number of vertices of G adjacent to vertices of P is minimized. The name secluded comes from the setting where one wants to transfer a confident information over a path in a network which can be intercepted either while passing through a vertex of the path or from some adjacent vertex. Thus the problem is to select a secluded path minimizing the risk of interception of the information. When instead of connecting two vertices one needs to connect a set of terminals, we arrive naturally to the Secluded Steiner Tree.

More precisely, Secluded Steiner Tree is the following problem.

Secluded Steiner Tree
Input: A graph G with a cost function $\omega\colon V(G) \to \mathbb{N}$, a set $S = \{s_1, \ldots, s_p\} \subseteq V(G)$ of terminals, and non-negative integers k and C.
Question: Is there a connected subgraph T of G with $S \subseteq V(T)$ such that $|N_G[V(T)]| \leq k$ and $\omega(N_G[V(T)]) \leq C$?

If $\omega(v) = 1$ for each $v \in V(G)$ and $C = k$, then we have an instance of Secluded Steiner Tree without costs; respectively, we omit ω and C whenever we consider such instances.

Clearly, it can be assumed that T is a tree, and thus the problem can be seen as a variant of the classical Steiner Tree problem. For the special case $p = 2$, we call the problem Secluded Path.

Previous work. The study of the secluded connectivity was initiated by Chechik et al. [7, 8] who showed that the decision version of Secluded Path without costs is NP-complete. Moreover, for the optimization version of the problem, it is hard to approximate within a factor of $\mathcal{O}(2^{\log^{1-\varepsilon} n})$, n is the number of vertices in the input graph, for any $\varepsilon > 0$ (under an appropriate complexity assumption) [8]. Chechik et al. [8] also provided several approximation and parameterized algorithms for Secluded Path and Secluded Steiner Tree. Interestingly, when there are no costs, Secluded Path is solvable in time $\Delta^{\Delta} \cdot n^{\mathcal{O}(1)}$, where Δ is the maximum vertex degree and and thus is FPT being parameterized by Δ. Chechik et al. [8] also showed that when the treewidth of the input graph does not exceed t, then the Secluded Steiner Tree problem is solvable in time $2^{\mathcal{O}(t \log t)} \cdot n^{O(1)} \cdot \log W$,[1] where W is the maximum value of ω on an input graph G. Johnson et al. [18] obtained several approximation results for Secluded Path and showed that the problem with costs is NP-hard for subcubic graphs improving the previous result of Chechik et al. [8] for graphs of maximum degree 4.

The problems related to secluded path and connectivity under different names were considered by several authors. Motivated by secure communications in wireless ad hoc networks, Gao et al.[15] introduced the very similar notion of the thinnest path. The motivation of Gilbers [17], who introduced the problem under the name of the minimum witness path, came from the study of art gallery problems.

Our results. In this paper we initiate the systematic study of both problems from the Parameterized Complexity perspective and obtain the following results. In Section 3, we start from observations that Secluded Path and Secluded Steiner Tree are FPT when parameterized by the size of the solution k by giving algorithms of running time

[1] In fact, Chechik et al. [8] give the algorithm that finds a tree with the exposure of minimum cost, but the algorithm can be easily modified for the more general Secluded Steiner Tree.

$\mathcal{O}(3^{k/3} \cdot (n+m) \log W)$ and $\mathcal{O}(2^k k^2 \cdot (n+m) \log W)$, where W is the maximum value of ω on an input graph G, correspondingly.

We consider the "above guarantee" parameterizations of both problems in Section 4. Recall that if $s_1, \ldots, s_p$ are vertices of a graph G, then a connected subgraph T of G of minimum size such that $s_1, \ldots, s_p \in V(T)$ is called a *Steiner tree* for the terminals $s_1, \ldots, s_p$. If $p = 2$, then a Steiner tree is a shortest (s_1, s_2)-path. Clearly, if ℓ is the size (the number of vertices) of a Steiner tree, then for any connected subgraph T of G with $S \subseteq V(T)$, $|N_G[V(T)]| \geq \ell$. Recall that the Steiner Tree problem is well known to be NP-complete as it is included in the famous Karp's list of 21 NP-complete problems [19], but in 1971 Dreyfus and Wagner [12] proved that the problem can be solved in time $O^*(3^p)$, i.e., it is FPT when parameterized by the number of terminals. The currently best FPT-algorithms for Steiner Tree running in time $O^*(2^p)$ are given by Björklund et al. [2] and Nederlof [21] (the first algorithm demands exponential in p space and the latter uses polynomial space). In Section 4 we show that Secluded Path and Secluded Steiner Tree are FPT when the problems are parameterized by $r + p$, where $r = k - \ell$. From the other side, we show that the problem is co-W[1]-hard when parameterized by r only.

In Section 5, we provide a thorough study of the kernelization of the problem from the structural paramaterization perspective. We consider parameterizations by the treewidth, size of the solution, maximum degree and the size of a vertex cover of the input graph. We show that it is unlikely that Secluded Path (even without costs) parameterized by the solution size, the treewidth and the maximum degree of the input graph, admits a polynomial kernel. In particular, this complements the FPT algorithmic findings of Chechik et al. [8] for graphs of bounded treewidth and of bounded maximum vertex degree. The same holds for the "above guarantee" parameterization instead the solution size as well. On the other hand, we show that Secluded Steiner Tree has a polynomial kernel when parameterized by k and the vertex cover number of the input graph. Interestingly, when we parameterize only by the vertex cover number, again, we show that most likely the problem does not admit a polynomial kernel. Finally, we refine the algorithm on graphs of bounded treewidth of Chechik et al. [8] by showing that Secluded Steiner Tree without costs can be solved by a randomized algorithm in time that single-exponentially depends on treewidth by applying the Count & Color technique of Cygan et al. [10] and further observe that for the general variant of the problem with costs, the same Count & Color technique can be used as well and also a single-exponential deterministic algorithm can be obtained by making use the representative set technique developed by Fomin et al. [14].

Due to space restrictions some proofs are omitted in this extended abstract. The full version of the paper is available in [13].

2 Basic definitions and preliminaries

We consider only finite undirected graphs without loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and the edge set is denoted by $E(G)$. Throughout the paper we typically use n and m to denote the number of vertices and edges respectively.

For a set of vertices $U \subseteq V(G)$, $G[U]$ denotes the subgraph of G induced by U. For a vertex v, we denote by $N_G(v)$ its *(open) neighborhood*, that is, the set of vertices which are adjacent to v, and for a set $U \subseteq V(G)$, $N_G(U) = (\cup_{v \in U} N_G(v)) \setminus U$. The *closed neighborhood* $N_G[v] = N_G(v) \cup \{v\}$. Respectively, $N_G[U] = N_G(U) \cup U$. For a set $U \subseteq V(G)$, $G - U$ denotes the subgraph of G induced by $V(G) \setminus U$. If $U = \{u\}$, we write $G - u$ instead of $G - \{u\}$. The *degree* of a vertex v is denoted by $d_G(v) = |N_G(v)|$. We say that a vertex

v is *pendant* if $d_G(v) = 1$. A vertex v of a connected graph G with at least 2 vertices is a *cut vertex* if $G - u$ is disconnected. A connected graph G is *biconnected* if it has at least 2 vertices and has no cut vertices. A *block* of a connected graph G is an inclusion-maximal biconnected subgraph of G. A block is *trivial* if it has exactly 2 vertices. We say that vertex set X is connected if $G[X]$ is connected.

Parameterized complexity is a two dimensional framework for studying the computational complexity of a problem. One dimension is the input size n and another one is a parameter k. It is said that a problem is *fixed parameter tractable* (or FPT), if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function f. A *kernelization* for a parameterized problem is a polynomial algorithm that maps each instance (x, k) with the input x and the parameter k to an instance (x', k') such that i) (x, k) is a yes-instance if and only if (x', k') is a yes-instance of the problem, and ii) the size of x' is bounded by $f(k)$ for a computable function f. The output (x', k') is called a *kernel*. The function f is said to be a *size* of a kernel. Respectively, a kernel is *polynomial* if f is polynomial. While a parameterized problem is FPT if and only if it has a kernel, it is widely believed that not all FPT problems have polynomial kernels. In particular, Bodlaender et al. [4, 5] introduced techniques that allow to show that a parameterized problem has no polynomial kernel unless NP $\subseteq$ coNP /poly. We refer to the recent books of Cygan et al. [9] and Downey and Fellows [11] for detailed introductions to parameterized complexity.

We use randomized algorithms for our problems. Recall that a *Monte Carlo algorithm* is a randomized algorithm whose running time is deterministic, but whose output may be incorrect with a certain (typically small) probability. A Monte-Carlo algorithm is *true-biased* (*false-biased* respectively) if it always returns a correct answer when it returns a yes-answer (a no-answer respectively).

3 FPT-algorithms for the problems parameterized by the solution size

In this section we consider Secluded Path and Secluded Steiner Tree problems parameterized by the size of the solution, i.e., by k. We also show how these parameterized algorithms can be used to design faster exact exponential algorithms.

We start with Secluded Path.[2]

▶ **Theorem 1** (∗). Secluded Path *is solvable in time* $\mathcal{O}(3^{k/3} \cdot n \log W)$*, where* W *is the maximum value of* ω *on an input graph* G*.*

For Secluded Steiner Tree we prove the following theorem.

▶ **Theorem 2** (∗). Secluded Steiner Tree *can be solved in time* $\mathcal{O}(2^k k^2 \cdot (n+m) \log W)$*, where* W *is the maximum value of* ω *on an input graph* G*.*

Parameterized algorithms for Secluded Path and Secluded Steiner Tree combined with a brute-force procedure imply the following exact exponential algorithms for the problems.

▶ **Theorem 3** (∗). *On an* n*-vertex graph,* Secluded Path *is solvable in time* $\mathcal{O}(1.3896^n \cdot \log W)$ *and* Secluded Steiner Tree *is solvable in time* $\mathcal{O}(1.7088^n \cdot \log W)$*, where* W *is the maximum value of* ω *on an input graph* G*.*

[2] The proofs of statements marked with (∗) can be found in [13].

4 FPT-algorithms for the problems parameterized above the guaranteed value

In this section we show that Secluded Path and Secluded Steiner Tree are FPT when the problems are parameterized by $r+p$ where $r=k-\ell$ and ℓ is the size of a Steiner tree for S.

▶ **Theorem 4** (∗). Secluded Path *is solvable in time* $\mathcal{O}(2^{k-\ell}\cdot(n+m)\log W)$*, where* ℓ *is the length of a shortest* (u,v)*-path for* $\{u,v\}=S$ *and* W *is the maximum value of* ω *on an input graph* G*.*

We need some structural properties of solutions of Secluded Steiner Tree. We start with an auxiliary lemma bounding the number of vertices of degree at least three in the subgraph of G induced by a solution as well as the number of their neighbors in this subgraphs.

▶ **Lemma 5.** *Let* G *be a connected graph and* $S\subseteq V(G)$*,* $p=|S|$*. Let* F *be an inclusion minimal connected induced subgraph of* G *such that* $S\subseteq V(F)$ *and* $X=\{v\in V(F)|d_F(v)\geq 3\}\cup S$*. Then (i)* $|X|\leq 4p-6$*, and (ii)* $|N_F(X)|\leq 4p-6$*.*

Proof. Let $\mathcal{B}$ be the set of blocks of F. Consider a bipartite graph T with the bipartition $(V(F),\mathcal{B})$ of the vertex such that $v\in V(F)$ and $b\in\mathcal{B}$ are adjacent if and only if v is a vertex of b. Notice that T is a tree. Recall that the *vertex dissolution* operation for a vertex v of degree 2 deletes v together with incident edges and replaces them by the edge joining the neighbors of v. Denote by T' the tree obtained from T by consequent dissolving all vertices of T of degree 2 that are not in S. Denote by L the set of leaves of T. By the minimality of F, $L\subseteq S$. Let $q_1=|L|\leq p$, and let q_2 be the number of degree 2 vertices and q_3 be the number of vertices of degree at least 3 in T. Clearly, $q_1+2q_2+3q_3\leq 2|E(T)|=2(q_1+q_2+q_3-1)$. Then $q_3\leq q_1-2\leq p-2$. We have that $|\{v\in V(T)|d_T(v)\geq 3\}\cup S|\leq q_3+p\leq 2p-2$ and $|V(T')|\leq 2p-2$. Observe that if $d_F(v)\geq 3$ for $v\in V(F)\setminus S$, then v is a cut vertex of F and either v is included in at least 3 blocks of F, or v is in a block of size at least 3. In the second case, v is adjacent to a vertex $b\in\mathcal{B}$ of T with degree at least 3. It implies that $|X|\leq 2|E(T')|=2(|V(T')|-1)\leq 4p-6$ and we have (i). To show (ii), observe that $|N_F(X)|\leq 2|E(T')|\leq 4p-6$. ◀

The following lemma provides a bound on the number of vertices of a tree that have neighbors outside the tree.

▶ **Lemma 6.** *Let* G *be a connected graph and* $S\subseteq V(G)$*,* $p=|S|$*. Let* ℓ *be the size of a Steiner tree for* S *and* r *be a positive integer. Suppose that* T *is an inclusion minimal subgraph of* G *such that* T *is a tree spanning* S *and* $|N_G[V(T)]|\leq\ell+r$*. Then for* $Y=N_G(V(T))$*,* $|N_G(Y)\cap V(T)|\leq 4p+2r-5$*.*

Proof. Denote by L the set of leaves of T and by D the set of vertices of degree at least 3 in T. Clearly, $L\subseteq S$. We select a leaf z of T as the *root* of T. The selection of a root defines a parent-child relation on T. We order the vertices of T by the increase of their distances to z in T; the vertices on the same distance are ordered arbitrarily. Denote the obtained linear order by $\prec$. For each $u\in Y$, denote by $x(u)$ the unique minimum vertex in $N_G(u)\cap V(T)$ with respect to $\prec$. Let $U=\{x(u)|u\in Y\}$. For a vertex $u\in Y$ and $v\in N_G(u)\cap V(T)\setminus\{x(u)\}$, let $y(u,v)$ be the parent of v in T. Let $W=\{y(u,v)|u\in Y,v\in N_G(u)\cap V(T),v\neq x(u)\}$ and $W'=W\setminus(S\cup D\cup U)$.

Let $F=G[V(T)\cup Y]$.

▶ **Claim 7.** *Set $F' = F - W'$ is connected.*

Proof of the claim. Since all leaves of T including z are in S, we have that $z \in V(F')$. To prove the claim, we show that for each vertex $v \in V(F')$, there is a (v,z)-path in F'. Every vertex $u \in Y$ has a neighbor $x(u)$ in F'. Hence, it is sufficient to prove the existence of (v,z)-paths for $v \in V(T) \setminus W'$. The proof is by induction on the number of v with respect to $\prec$. The first vertex is z, and if $v = z$, then we have a trivial (z,v)-path. Assume that $v \neq z$. Let w be the parent of v in T. If $w \in V(F')$, then $w \prec v$ and, by the inductive hypothesis, there is a (z,w)-path in F' and it implies the existence of a (z,v)-path. Suppose that $w \notin V(F')$, i.e., $w \in W'$. Since $d_T(w) = 2$, there is $u \in Y$ such that $w = y(u,v)$. We have that $x(u) \prec v$ and there is a $(z,x(u))$-path in F' by the inductive hypothesis. It remains to observe that because $x(u)u, uv \in E(F')$, F' has a (z,v)-path as well. This concludes the proof to the claim. ◀

Denote by R the set of the children of the vertices of $D \cup S$ in T. Observe that $|N_G(Y) \cap V(T)| \leq |D \cup S| + |R| + |U| + |W'|$. Recall that $|V(F)| \leq \ell + r$. Because F' is connected and $S \subseteq V(F')$, $|V(F')| \geq \ell$. Hence, $|W'| \leq r$. Let $q_1 = |L|$, $q_2 = |V(T) \setminus (L \cup D)|$ and $q_3 = |D|$. We have that $q_1 + 2q_2 + 3q_3 \leq 2|E(T)| = 2(q_1 + q_2 + q_3 - 1)$. Then $q_3 \leq q_1 - 2$ and $|D \cup S| \leq 2|S| - 2 = 2p - 2$, because $L \subseteq S$. Let T' be the tree obtained from T by consequent dissolving all the vertices of degree 2 that are not in S. Then $|R| \leq |E(T')| \leq 2|S| - 3 = 2p - 3$. Since $|V(T)| \geq \ell$, $|U| \leq |Y| \leq r$. We obtain that $|N_G(Y) \cap V(T)| \leq |D \cup S| + |R| + |U| + |W'| \leq 2p - 2 + 2p - 3 + r + r = 4p + 2r - 5$. ◀

Now we are ready to prove the main result of the section.

▶ **Theorem 8.** Secluded Steiner Tree *can be solved in time $2^{O(p+r)} \cdot nm \cdot \log W$ by a true-biased Monte-Carlo algorithm and in time $2^{O(p+r)} \cdot nm \log n \cdot \log W$ by a deterministic algorithm for graphs with n vertices and m edges, where $r = k - \ell$ and ℓ is the size of a Steiner tree for S and W is the maximum value of ω on an input graph G.*

Proof. We construct an FPT-algorithm for Secluded Steiner Tree parameterized by $p + r$. The algorithm is based on the random separation techniques introduced by Cai, Chan, and Chan [6] (see also [1]). We first describe a randomized algorithm and then explain how it can be derandomized.

Let $\mathcal{I} = (G, \omega, S, k, C)$ be an instance of Secluded Steiner Tree, ℓ be the size of a Steiner tree for $S = \{s_1, \ldots, s_p\}$ and $r = k - \ell$. Without loss of generality we assume that $p \geq 2$ and $r \geq 1$ as for $p = 1$ or $r = 0$, the problem is trivial. We also can assume that G is connected.

Description of the algorithm. In each iteration of the algorithm we color the vertices of G independently and uniformly at random by two colors. In other words, we partition $V(G)$ into two sets R and B. We say that the vertices of R are *red*, and the vertices of B are *blue*. Our algorithm can recolor some blue vertices red, i.e., the sets R and B can be modified. Our aim is to find a connected subgraph T of G with $S \subseteq V(T)$ such that $|N_G[V(T)]| \leq k$, $\omega(N_G[V(T)]) \leq C$ and $V(T) \subseteq R$.

Step 1. If $G[R]$ has a component H such that $S \subseteq V(H)$, then find a spanning tree T of H. If $|N_G[V(T)]| \leq k$ and $\omega(N_G[V(T)]) \leq C$, then return T and stop; otherwise, return that $\mathcal{I}$ is no-instance and stop.

Step 2. If there is $s_i \in S$ such that $s_i \notin R$ or $N_G(s_i) \cap R = \emptyset$, then return that $\mathcal{I}$ is no-instance and stop.

Step 3. Find a component H of $G[R]$ with $s_1 \in V(H)$. If there is a pendant vertex $u \notin S$ of H that is adjacent in G to a unique vertex $v \in B$, then find a component of $G[B]$ that contains v, recolor its vertices red and then return to Step 1. Otherwise, return that (G, S, k) is no-instance and stop.

We repeat at most $2^{O(r+p)}$ iterations. If on some iteration we obtain a yes-answer, then we return it and the corresponding solution. Otherwise, if on every iteration we get a no-answer, we return a no-answer.

Correctness of the algorithm. It is straightforward to see that if this algorithm returns a tree T in G with $|N_G[V(T)]| \leq k$ and $\omega(N_G[V(T)]) \leq C$, then we have a solution for the considered instance of Secluded Steiner Tree. We show that if $\mathcal{I}$ is a yes-instance, then there is a positive constant α that does not depend on n and r such that the algorithm finds a tree T in G with $|N_G[V(T)]| \leq k$ and $\omega(N_G[V(T)]) \leq C$ with probability at least α after $2^{O(p+r)}$ executions of this algorithm for random colorings.

Suppose that $\mathcal{I}$ is a yes-instance. Then there is a tree T in G such that $S \subseteq V(T)$, $|N_G[V(T)]| \leq k$ and $\omega(N_G[V(T)]) \leq C$. Without loss of generality we assume that T is inclusion minimal. Let $F = G[V(T)]$, $X = \{v \in V(F) | d_F(v) \geq 3\} \cup S$, $X' = N_F(X)$, $Y = N_G(V(T))$ and $Y' = N_G(Y) \cap V(T)$. For each $v \in Y' \setminus S$, we arbitrarily select two distinct neighbors $z_1(v)$ and $z_2(v)$ in T. Because the leaves of T are in S, we have that v is not a leaf and thus has at least two neighbors. Let $Z = \{z_i(v) | v \in Y' \setminus S, i = 1, 2\}$. Let $W = X \cup X' \cup Y \cup Y' \cup Z$.

By Lemma 5, $|X| \leq 4p-6$ and $|X'| \leq 4p-6$. By Lemma 6, $|Y'| \leq 4p+2r-5$ and, therefore, $|Z| \leq 8p+4r-10$. Because $|V(T)| \geq \ell$ and $|N_G[V(T)]| \leq \ell + r$, we have that $|Y| \leq r$. Hence $|W| \leq |X|+|X'|+|Y|+|Y'|+|Z| \leq 4p-6+4p-6+r+4p+2r-5+8p+4r-10 = 20p+7r-27$. Let $N = 20p+7r-27$. Then with probability at least 2^{-N}, the vertices of Y are colored blue and the vertices of $X \cup X' \cup Y' \cup Z$ are colored red, i.e., $W \cap V(T) \subseteq R$ and $W \setminus V(T) \subseteq B$. The probability that for a random coloring, the vertices of W are colored incorrectly, i.e., $W \cap V(T) \cap B \neq \emptyset$ or $(W \setminus V(T)) \cap R \neq \emptyset$, is at most $1 - 2^{-N}$. Hence, if we consider 2^N random colorings, then the probability that the vertices of W are colored incorrectly for all the colorings is at most $(1 - 2^{-N})^{2^N}$, and with probability at least $1 - (1 - 2^{-N})^{2^N}$ for at least one coloring we will have $W \cap V(T) \subseteq R$ and $W \setminus V(T) \subseteq B$. Since $(1 - 2^{-N})^{2^N} \leq 1/e$, we have that $1 - (1 - 2^{-N})^{2^N} \leq 1 - 1/e$. Thus if $\mathcal{I}$ is a yes-instance, after 2^N random colorings of G, we have that at least one of the colorings is successful with a constant success probability $\alpha = 1 - 1/e$.

Assume that for a random red-blue coloring of G, $W \cap V(T) \subseteq R$ and $W \setminus V(T) \subseteq B$. We show that in this case the algorithm finds a tree T' with $S \subseteq V(T') \subseteq V(T)$. Clearly, $|N_G[V(T')]| \leq |N_G[V(T)]| \leq k$ and $\omega(N_G[V(T')]) \leq \omega(N_G[V(T)]) \leq C$ in this case.

We claim that for every connected component H of $G[R]$, either $V(H) \subseteq V(T)$ or $V(H) \cap V(T) = \emptyset$. To obtain a contradiction, assume that there are $u, v \in V(H)$ such that $u \in V(T)$ and $v \notin V(T)$. Indeed, H is connected, and thus contains an (u, v) path P. Since P goes from $V(T)$ to $v \notin V(T)$, path P should contain a vertex $w \in N_G(T) = Y$. But w is colored blue, which is a contradiction to the assumption that P is in the red component H. By the same arguments, for any component H of $G[B]$, either $V(H) \subseteq V(T)$ or $V(H) \cap V(T) = \emptyset$.

We consider Steps 1–3 of the algorithm and show their correctness.

Suppose that $G[R]$ has a component H such that $S \subseteq V(H)$. Because $S \subseteq W \cap V(T) \subseteq R$, $V(H) \cap V(T) \neq \emptyset$ and, therefore, $V(H) \subseteq V(T)$. Then for every spanning tree T' of H, $S \subseteq V(T')$ and $N_G[V(T')] \subseteq N_G[V(T)]$. Therefore, $|N_G[V(T')]| \leq |N_G[V(T)]| \leq k$ and $\omega(N_G[V(T')]) \leq \omega(N_G[V(T)]) \leq C$. Hence, if a component of $G[R]$ contains S, then we find a solution. This concludes the proof of the correctness of the first step.

Let us assume that the algorithm does not stop at Step 1. For the right coloring, because $S \subseteq X$ and $N_F(S) \subseteq X'$, for every $s_i \in S$, we have that $s_i \in R$. Moreover, because $p \geq 2$, at least one neighbor of s_i in G is in R. Thus the only reason why the algorithm stops at Step 2 is due to the wrong coloring. Consider the case when the algorithm does not stop after Step 2.

Suppose that H is a component of $G[R]$ with $s_1 \in V(H)$. Because the algorithm did not stop in Step 2, such a component H exists and has at least 2 vertices. Recall that $V(H) \subseteq V(T)$. Because we proceed in Step 1, we conclude that $S \setminus V(H) \neq \emptyset$. Then there is a vertex $u \in V(H)$ which has a neighbor v in T such that $v \in B$. If $u \in S$, then $v \in X'$, but this contradicts the assumption $X' \subseteq R$. Hence, $u \notin S$. Suppose that $d_H(u) \geq 2$. In this case $d_F(u) \geq 3$ and $v \in X'$; a contradiction. Therefore, u is a pendant vertex of H.

Let $u \notin S$ be an arbitrary pendant vertex of H. If u has no neighbors in B, then u is a leaf of T that does not belong to S but this contradicts the inclusion minimality of T. Assume that u is adjacent to at least two distinct vertices of B. Because T is an inclusion minimal tree spanning S, vertex u has at least two neighbors in T and u has a neighbor $v \in B$ in T. Let $w \in (N_G(u) \cap B) \setminus \{v\}$. If $w \in V(T)$, then $d_F(u) \geq 3$ and, therefore, $u \in X$ and $v, w \in X'$; a contradiction with $X' \subseteq R$. Hence, $w \notin V(T)$. Moreover, v is the unique neighbor of u in T that belongs to B. Then $w \in Y$ and $v \in \{z_1(u), z_2(u)\}$; a contradiction with $Z \subseteq R$. We obtain that u is adjacent in G to a unique vertex $v \in B$. Let H' be the component of $G[B]$ that contains v. Since T is an inclusion minimal tree that spans S, u has at least two neighbors in T. It implies that $v \in V(T)$, therefore $V(H') \subseteq V(T)$. We recolor the vertices of H' red in Step 3. For the new coloring the vertices of Y are blue and the vertices of $W \setminus Y$ are red. Therefore, we keep the crucial property of the considered coloring but we increase the size of the component of $G[R]$ containing s_1.

To conclude the correctness proof, it remains to observe that in Step 3 we increase the number of vertices in the component of $G[R]$ that contains s_1. Hence, after at most n repeats of Steps 1-3, we obtain a component in $G[R]$ that includes S and return a solution in Step 1.

It is straightforward to verify that each of Steps 1–3 can be done in time $O(m \log W)$. Because the number of iterations is at most n, we obtain that the total running time is $2^{O(p+r)} \cdot nm \log W$.

This algorithm can be derandomized by standard techniques (see [1, 6]). The random colorings can be replaced by the colorings induced by *universal sets*. Let n and q be positive integers, $q \leq n$. An (n, q)-*universal set* is a collection of binary vectors of length n such that for each index subset of size q, each of the 2^q possible combinations of values appears in some vector of the set. It is known that an (n, q)-universal set can be constructed in FPT-time with the parameter q. The best construction is due to Naor, Schulman and Srinivasan [20]. They obtained an (n, q)-universal set of size $2^q \cdot q^{O(\log q)} \log n$, and proved that the elements of the sets can be listed in time that is linear in the size of the set. In our case n is the number of vertices of G and $q = 20p + 7r - 27$. ◀

We complement Theorem 8 by showing that it is unlikely that Secluded Steiner Tree is FPT if parameterized by r only. To show it, we use the standard reduction from the Set Cover problem (see, e.g., [19]). Notice that we prove that Secluded Steiner Tree is co-W[1]-hard, i.e., we show that it is W[1]-hard to decide whether we have a no-answer.

▶ **Theorem 9** (∗). Secluded Steiner Tree *without costs is co-*W[1]*-hard when parameterized by r, where $r = k - \ell$ and ℓ is the size of a Steiner tree for S.*

5 Structural parameterizations of Secluded Steiner Tree

In this section we consider different algorithmic and complexity results concerning different structural parameterizations of secluded connectivity problems. We consider parameterizations by the treewidth, size of the solution, maximum degree and the size of a vertex cover of the input graph (see [13] for definitions of these parameters.) We show that under reasonable complexity assumptions Secluded Path without costs has no polynomial kernel when parameterized by $k + t + \Delta$, where t is the treewidth and Δ is the maximum degree of the input graph. We obtain the same result for the cases when the problem is parameterized by $k - \ell + t + \Delta$, where ℓ is the length of the shortest path between terminals.

▶ **Theorem 10** (∗). Secluded Path *without costs on graphs of treewidth at most t and maximum degree at most Δ admits no polynomial kernel unless* NP $\subseteq$ coNP /poly *when parameterized by $k + t + \Delta$ or $(k - \ell) + t + \Delta$, where ℓ is the length of the shortest path between terminals.*

Observe that Theorem 10 immediately implies that Secluded Path without costs has no polynomial kernel unless NP $\subseteq$ coNP /poly when parameterized by k or $k - \ell$. The next natural question is if parameterization by a stronger parameter can lead to a polynomial kernel. Let us note that the treewidth of a graph is always at most the minimum size of its vertex cover. The following theorem provides lower bounds for parameterization by the minimum size of a vertex cover.

▶ **Theorem 11.** Secluded Path *without costs on graphs with the vertex cover number at most w has no polynomial kernel unless* NP $\subseteq$ coNP /poly *when parameterized by w.*

Proof. The proof uses the cross-composition technique introduced by Bodlaender, Jansen and Kratsch [5]. We show that the 3-Satisfiability problem OR-cross composes into Secluded Path without costs. Recall that 3-Satisfiability asks for given boolean variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ with 3 literals each, whether the formula $\phi = C_1 \wedge \ldots \wedge C_m$ can be satisfied. It is well-known that 3-Satisfiability is NP-complete [16]. We assume that two instances of 3-Satisfiability are equivalent if they have the same number of variables and the same number of clauses.

Consider t equivalent instances of 3-Satisfiability with the same boolean variables $x_1, \ldots, x_n$ and the sets of clauses $\mathcal{C}_i = \{C_1^i, \ldots, C_m^i\}$ for $i \in \{1, \ldots, t\}$. Without loss of generality we assume that $t = \binom{2q}{q}$ for a positive integer q; otherwise, we add minimum number of copies of $\mathcal{C}_1$ to get this property. Notice that $\binom{2q}{q} = \Theta(4^q/\sqrt{\pi q})$ and $q = O(\log t)$. Let $I_1, \ldots, I_t$ be pairwise distinct subsets of $\{1, \ldots, 2q\}$ of size q. Notice that each $i \in \{1, \ldots, 2q\}$ is included exactly in $d = \binom{2q-1}{q-1}$ sets. Let $k = (q + 3d)m + 3q + 4n + 2$. We construct the graph G as follows (see Fig. 1).

- **(i)** Construct $n + 1$ vertices $u_0, \ldots, u_n$. Let $s_1 = u_0$.
- **(ii)** For each $i \in \{1, \ldots, n\}$, construct vertices $x_i, y_i, \overline{x}_i, \overline{y}_i$ and edges $u_{i-1}y_i, y_iu_i, y_ix_i$, and $u_{i-1}\overline{y}_i, \overline{y}_iu_i, \overline{y}_i\overline{x}_i$.
- **(iii)** For each $j \in \{0, \ldots, m\}$, construct a set of vertices $W_j = \{w_1^j, \ldots, w_{2q}^j\}$.
- **(iv)** Construct a vertex s_2 and edges $u_nw_1^0, \ldots, u_nw_{2q}^0$ and $w_1^ms_2, \ldots, w_{2q}^ms_2$.
- **(v)** For each $j \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, t\}$,
  - construct 3 vertices $c_{jh}^1, c_{jh}^2, c_{jh}^3$;

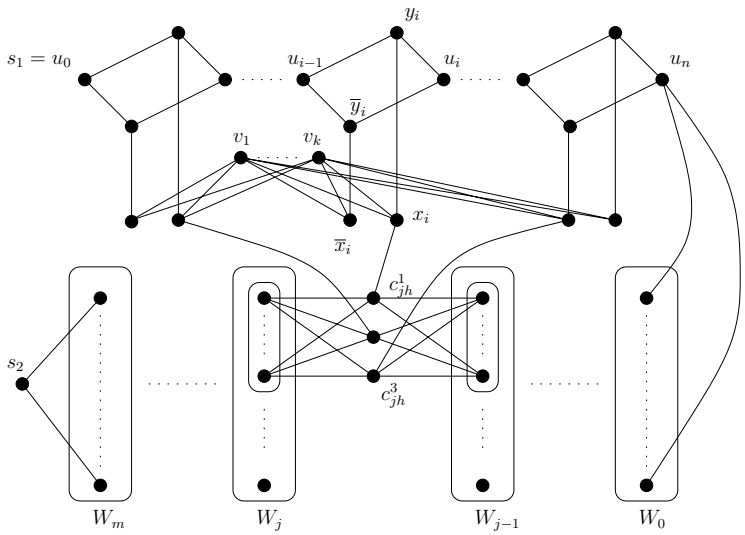


Figure 1 Construction of G.

- construct edges $c^1_{jh}w^{j-1}_r, c^2_{jh}w^{j-1}_r, c^3_{jh}w^{j-1}_r$ and $c^1_{jh}w^j_r, c^2_{jh}w^j_r, c^3_{jh}w^j_r$ for all $r \in I_h$;
- consider the clause $C^h_j = (z_1 \vee z_2 \vee z_3)$ and for $l \in \{1,2,3\}$, construct an edge $c^l_{jh}x_i$ if $z_l = x_i$ for some $i \in \{1,\ldots,n\}$ and construct an edge $c^l_{jh}\overline{x}_i$ if $z_l = \overline{x}_i$.

(vi) Construct k vertices $v_1,\ldots,v_k$ and edges $x_iv_l, \overline{x}_iv_l$ for $i \in \{1,\ldots,n\}$ and $l \in \{1,\ldots,k\}$. Observe that the set of vertices

$$X = (\cup^n_{i=1}\{x_i, y_i, \overline{x}_i, \overline{y}_i\}) \cup (\cup^m_{j=0}W_j)$$

is a vertex cover in G of size $4n + 2q(m+1) = O(n + m\log t)$.

We show that G has an (s_1,s_2)-path P with $|N_G[V(P)]| \leq k$ if and only if there is $h \in \{1,\ldots,t\}$ such that $x_1,\ldots,x_n$ have a truth assignment satisfying all the clauses of $\mathcal{C}_h$.

Suppose that $x_1,\ldots,x_n$ have an assignment that satisfies all the clauses of $\mathcal{C}_h$. First, we construct the (s_1,u_n)-path P' by the concatenation of the following paths: for each $i \in \{1,\ldots,n\}$, we take the path $u_{i-1}y_iu_i$ if $x_i = true$ in the assignment and we take $u_{i-1}\overline{y}_iu_i$ if $x_i = false$. Let $r \in I_h$. We construct the (w^0_r, w^m_r)-path P'' by concatenating $w^{j-1}_r c^{l_j}_{jh} w^j_r$ for $j \in \{1,\ldots,m\}$ where $l_j \in \{1,2,3\}$ is chosen as follows. Each clause $C^h_j = z_1 \vee z_2 \vee z_3 = true$ for the assignment, i.e., $z_l = true$ for some $l \in \{1,2,3\}$; we set $l_j = l$. Finally, we set $P = P' + u_nw^0_h + P'' + w^m_hs_2$. It is straightforward to verify that $|N_G[V(P)]| = k$.

Suppose now that there is an (s_1,s_2)-path in G with $|N_G[V(P)]| \leq k$. We assume that P is an induced path. Observe that $x_i, \overline{x_i} \notin V(P)$ for $i \in \{1,\ldots,n\}$, because $d_G(x_i), d_G(\overline{x}_i) > k$. Therefore, P has an (s_1,u_n)-subpath P' such that $u_0,\ldots,u_n \in V(P')$ and for each $i \in \{1,\ldots,n\}$, either $y_i \in V(P')$ or $\overline{y}_i \in V(P')$. We set the variable $x_i = true$ if $x_i \in V(P')$ and $x_i = false$ otherwise. We show that this truth assignment satisfies all the clauses of some $\mathcal{C}_h$.

Observe that $|N_G[V(P')]| = 4n + 2q + 1$. Clearly, $s_2 \in V(P)$. Notice also that P has at least one vertex in each W_j for $j \in \{0,\ldots,m\}$, and for each $j \in \{1,\ldots,m\}$, at least one vertex among the vertices c^l_{jh} for $h \in \{1,\ldots,t\}$ and $l \in \{1,2,3\}$ is in P. For each $j \in \{1,\ldots,m\}$, any two verices $w^{j-1}_r \in W_{j-1}$ and $w^j_{r'} \in W_j$ have at least $3d$ neighbors among the vertices c^l_{jf} for $f \in \{1,\ldots,t\}$ and $l \in \{1,2,3\}$. Moreover, if $r \neq r'$, they have at least $3d+6$ such neighbors, because there are two subsets $I, I' \subseteq \{1,\ldots,2q\}$ of size q such

that $r \in I \setminus I'$ and $r' \in I' \setminus I$. For each $j \in \{1, \ldots, m-1\}$, any two vertices c^l_{jh} and $c^{l'}_{j+1\ h'}$ for $h, h' \in \{1, \ldots, t\}$ and $l, l' \in \{1, 2, 3\}$ have at least q neighbors in W_j. Moreover, if $h \neq h'$, they have at least $q+2$ such neighbors, because $|I_h \cup I_{h'}| \geq q+2$. Taking into account that $d_G(s_2) = 2q$, we obtain that

$$k \geq |N_G[V(P)]| \geq |N_G[V(P')]| + 3dm + q(m-1) + 2q + 1 = k.$$

It implies that P has exactly one vertex in each W_j for $j \in \{0, \ldots, m\}$, and for each $j \in \{1, \ldots, m\}$, exactly one vertex among the vertices c^l_{jh} for $h \in \{1, \ldots, t\}$ and $l \in \{1, 2, 3\}$ is in P. Moreover, there is $r \in \{1, \ldots, 2q\}$ and $h \in \{1, \ldots, t\}$ such that $w^j_r \in V(P)$ and $c^{l_j}_{jh} \in V(P)$ for $j \in \{0, \ldots, m\}$ and $l_j \in \{1, 2, 3\}$. We claim that all the clauses of $\mathcal{C}_r$ are satisfied. Otherwise, if there is a clause $C^r_j = (z_1 \vee z_2 \vee z_3)$ that is not satisfied, then the neighbors of $c^1_{jh}, c^2_{jh}, c^3_{jh}$ among the vertices $x_i, \overline{x}_i$ for $i \in \{1, \ldots, n\}$ are not in $N_G[V(P')]$. It immediately implies that $|N_G[V(P)]| > k$; a contradiction. ◀

However, if we consider even stronger parameterization, by vertex cover number and by the size of the solution, then we obtain the following theorem.

▶ **Theorem 12** (∗). *The* Secluded Steiner Tree *problem admits a kernel with at most* $2w(k+1)$ *vertices on graphs with the vertex cover number at most* w.

Recall that Chechik et al. [8] showed that if the treewidth of the input graph does not exceed t, then the Secluded Steiner Tree problem is solvable in time $2^{\mathcal{O}(t \log t)} \cdot n^{O(1)} \cdot \log W$, where W is the maximum value of ω on an input graph G. We observe that the running time could be improved by applying modern techniques for dynamic programming over tree decompositions proposed by Cygan et al. [10], Bodlaender et al. [3] and Fomin et al. [14]. Essentially, the algorithms for Secluded Steiner Tree are constructed along the same lines as the algorithms for Steiner Tree described in [10, 3, 14]. Hence, for simplicity, we only sketch the randomized algorithm based on the Cut&Count technique introduced by Cygan et al. [10] for Secluded Steiner Tree without costs in this conference version of our paper.

▶ **Theorem 13** (∗). *There is a true-biased Monte Carlo algorithm solving the* Secluded Steiner Tree *without costs in time* $4^t \cdot n^{\mathcal{O}(1)}$*, given a tree decomposition of width at most* t.

The algorithm based on the Cut&Count technique can be generalized for Secluded Steiner Tree with costs in the same way as the algorithm for Steiner Tree in [10]. This way we can obtain the algorithm that runs in time $4^t \cdot (n+W)^{\mathcal{O}(1)}$ where W is the maximal cost of vertices. One can obtain a deterministic algorithm and improve the dependence on W using the representative set technique for dynamic programming over tree decompositions introduced by Fomin et al. [14]. Again by the same approach as for Steiner Tree, it is possible to solve Secluded Steiner Tree deterministically in time $\mathcal{O}((2+2^{\omega+1})^t \cdot (n + \log W)^{\mathcal{O}(1)})$ (here ω is the matrix multiplication constant). We postpone the proof till the full version of the paper.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *STOC 2007*, pages 67–74. ACM, 2007.

3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *ICALP 2013, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2013.

4 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

6 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *IWPEC 2006*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006.

7 Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *CoRR*, abs/1212.6176, 2012.

8 Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. In *ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2013.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS 2011*, pages 150–159. IEEE, 2011.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

12 S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

13 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *CoRR*, abs/1502.03989, 2015.

14 Fedor V Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *SODA 2014*, pages 142–151, 2014.

15 Jianhang Gao, Qing Zhao, and Ananthram Swami. The thinnest path problem for secure communications: A directed hypergraph approach. In *Proceedings of the 50th Annual Allerton Conference on Communication, Control, and Computing, 2012*, pages 847–852. IEEE, 2012.

16 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

17 Alexander Gilbers. *Visibility Domains and Complexity*. PhD thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, 2013.

18 Matthew P. Johnson, Ou Liu, and George Rabanca. Secluded path via shortest path. In *SIROCCO 2014*, volume 8576 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2014.

19 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

20 M. Naor, L.J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *FOCS 1995*, pages 182–191. IEEE, 1995.

21 Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013.

Parameterized Algorithms for Deletion to (r, ℓ)-Graphs

Sudeshna Kolay and Fahad Panolan

Institute of Mathematical Sciences, Chennai, India
{skolay,fahad}@imsc.res.in

Abstract

For fixed integers $r, \ell \geq 0$, a graph G is called an (r, ℓ)-*graph* if the vertex set $V(G)$ can be partitioned into r independent sets and ℓ cliques. This brings us to the following natural parameterized questions: Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization. An input to these problems consist of a graph G and a positive integer k and the objective is to decide whether there exists a set $S \subseteq V(G)$ ($S \subseteq E(G)$) such that the deletion of S from G results in an (r, ℓ)-graph. These problems generalize well studied problems such as Odd Cycle Transversal, Edge Odd Cycle Transversal, Split Vertex Deletion and Split Edge Deletion. We do not hope to get parameterized algorithms for either Vertex (r, ℓ)-Partization or Edge (r, ℓ)-Partization when either of r or ℓ is at least 3 as the recognition problem itself is NP-complete. This leaves the case of $r, \ell \in \{1, 2\}$. We almost complete the parameterized complexity dichotomy for these problems by obtaining the following results:

1. We show that Vertex (r, ℓ)-Partization is fixed parameter tractable (FPT) for $r, \ell \in \{1, 2\}$. Then we design an $\mathcal{O}(\sqrt{\log n})$-factor approximation algorithms for these problems. These approximation algorithms are then utilized to design polynomial sized randomized Turing kernels for these problems.
2. Edge (r, ℓ)-Partization is FPT when $(r, \ell) \in \{(1, 2), (2, 1)\}$. However, the parameterized complexity of Edge $(2, 2)$-Partization remains open.

For our approximation algorithms and thus for Turing kernels we use an interesting finite forbidden induced graph characterization, for a class of graphs known as (r, ℓ)-split graphs, properly containing the class of (r, ℓ)-graphs. This approach to obtain approximation algorithms could be of an independent interest.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases FPT, Turing kernels, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.420

1 Introduction

For fixed integers $r, \ell \geq 0$, a graph G is called an (r, ℓ)-*graph* if the vertex set $V(G)$ can be partitioned into r independent sets and ℓ cliques. Although the problem has an abstract setting, some special cases are well known graph classes and have been widely studied. For example, $(2, 0)$- and $(1, 1)$-graphs correspond to bipartite graphs and split graphs respectively. A $(3, 0)$-graph is a 3-colourable graph. Already, we get a hint of an interesting dichotomy for this graph class, even with respect to recognition algorithms. Throughout the paper we will use m and n to denote the number of edges and the number of vertices, respectively, in the input graph G. It is well known that we can recognize $(2, 0)$- and $(1, 1)$-graphs in $\mathcal{O}(m + n)$ time. In fact, one can show that recognizing whether a graph G is an (r, ℓ)-graph, when $r, \ell \leq 2$, can be done in polynomial time [2, 9]. On the other hand, when either $r \geq 3$ or $\ell \geq 3$, the recognition problem is as hard as the celebrated 3-colouring problem, which

is NP-complete [12]. These problems are also studied when the input is restricted to be a chordal graph, in which case we can get polynomial time recognition algorithms for every r and ℓ [10].

The topic of this paper is to design recognition algorithms for *almost* (r, ℓ)-graphs in the realm of parameterized algorithms. In particular, we study the following natural parameterized questions on (r, ℓ)-graphs: Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization.

Vertex (r, ℓ)-Partization	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph?	

Edge (r, ℓ)-Partization	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Is there an edge subset $F \subseteq E(G)$ of size at most k such that $G - F$ is an (r, ℓ)-graph?	

These problems generalize some of the most well studied problems in parameterized complexity, such as Vertex Cover, Odd Cycle Transversal (OCT), Edge Odd Cycle Transversal (EOCT), Split Vertex Deletion (SVD) and Split Edge Deletion (SED). Vertex Cover, in particular, has been extensively studied in the parameterized complexity, and the current fastest algorithm runs in time $1.2738^k n^{\mathcal{O}(1)}$ and has a kernel with $2k$ vertices [4]. The parameterized complexity of OCT was a well known open problem for a long time. In 2003, in a breakthrough paper, Reed et al. [25] showed that OCT is FPT by developing an algorithm for the problem running in time $\mathcal{O}(3^k mn)$. In fact, this was the first time that the iterative compression technique was used. However, the algorithm for OCT had seen no further improvements in the last 9 years, though several reinterpretations of the algorithm have been published [16, 22]. Only recently, Lokshtanov et al. [21] obtained a faster algorithm for the problem running in time $2.3146^k n^{\mathcal{O}(1)}$ using a branching algorithm based on linear programming. Guo et al. [14] designed an algorithm for EOCT running in time $2^k n^{\mathcal{O}(1)}$. There is another theme of research in parameterized complexity, where the objective is to minimize the dependence of n at the cost of a slow growing function of k. A well known open problem, in the area, is whether OCT admits a linear time parameterized algorithms. Only recently, the first linear time FPT algorithms for OCT on general graphs were obtained, both of which run in time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$ [24, 17]. Kratsch and Wahlström [19] obtained a randomized polynomial kernel for OCT and EOCT. Ghosh et al. [13] studied SVD and SED and designed algorithms with running time $2^k n^{\mathcal{O}(1)}$ and $2^{\mathcal{O}(\sqrt{k}\log k)} n^{\mathcal{O}(1)}$. They also gave the best known polynomial kernel for these problems. Later, Cygan and Pilipczuk [7] designed an algorithm for SVD running in time $1.2738^{k+o(k)} n^{\mathcal{O}(1)}$. Krithika and Narayanaswamy [20] studied Vertex (r, ℓ)-Partization problems on perfect graphs, and among several results they obtain $(r+1)^k n^{\mathcal{O}(1)}$ algorithm for Vertex $(r, 0)$-Partization on perfect graphs.

Our Results and Methods. The instance of a parameterized problem is a pair containing the actual problem instance of size n and a positive integer called a parameter, usually represented as k. The problem is said to be in FPT if there exists an algorithm that solves the problem in $f(k)n^{\mathcal{O}(1)}$ time, where f is a computable function. An algorithm with such a running time is called an FPT algorithm. Readers are requested to refer [11] for more details. We do not hope to get FPT algorithms for either Vertex (r, ℓ)-Partization or Edge (r, ℓ)-Partization when either of r or ℓ is at least 3 as the recognition problem itself is

r, ℓ	Problem Name	FPT	Kernel
$(1, 0)$	Vertex Cover	1.2738^k	Poly
$(0, 1)$	Vertex Cover on $\overline{G}$	1.2738^k	Poly
$(1, 1)$	SVD	$1.2738^{k+o(k)}$	Poly
$(2, 0)$	OCT	2.3146^k	Randomized Poly
$(0, 2)$	OCT on $\overline{G}$	2.3146^k	Randomized Poly
$(2, 1)$, $(1, 2)$, $(2, 2)$	Vertex $(2, 1)$-partization Vertex $(1, 2)$-partization Vertex $(2, 2)$-partization	3.3146^k	**Randomized Turing Poly**

Figure 1 Summary of known and new results for the family of Vertex (r, ℓ)-Partization problems. New results are highlighted in green (last row).

r, ℓ	Problem Name	FPT	Kernel
$(1, 0)$	*Recognizable in polynomial time.*		
$(0, 1)$	*Recognizable in polynomial time.*		
$(1, 1)$	SED	$2^{\mathcal{O}(\sqrt{k} \log k)}$	Poly
$(2, 0)$	EOCT	2^k	Randomized Poly
$(0, 2)$	*Recognizable in polynomial time.*		
$(2, 1)$	Edge $(2, 1)$-partization	$2^{k+o(k)}$	**Open**
$(1, 2)$	Edge $(1, 2)$-partization	**FPT**	**Open**
$(2, 2)$	Edge $(2, 2)$-partization	**Open**	

Figure 2 Summary of known and new results for the family of Edge (r, ℓ)-Partization problems. New results are highlighted in green.

NP-complete. This leaves the case of $r, \ell \in \{0, 1, 2\}$. We almost complete the parameterized complexity dichotomy for these problems by either obtaining new results or using the existing results. We refer to Figures 1 and 2 for a summary of new and old results. Due to paucity of space, some proofs have had to be omitted from the paper. However, all such results (marked with a $\star$) have their complete proofs in the full version.[1]

For both Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization, the only new cases for which we need to design new parameterized algorithms to complete the dichotomy is when $r, \ell \in \{1, 2\}$. Apart from the algorithmic results indicated in the Figures 1 and 2, we also obtain the following results. When $r, \ell \in \{1, 2\}$, we obtain an $\mathcal{O}(\sqrt{\log n})$-approximation for these special cases. Finally, we obtain randomized *Turing kernels* for Vertex (r, ℓ)-Partization using this approximation algorithms. In other words, we give a polynomial time algorithm that produces polynomially many instances, $n^{\mathcal{O}(1)}$ of Vertex (r, ℓ)-Partization of size $k^{\mathcal{O}(1)}$ such that with very high probability (G, k) is a YES instance of Vertex (r, ℓ)-Partization if and only if one of the polynomially many instances of Vertex (r, ℓ)-Partization of size $k^{\mathcal{O}(1)}$ is a YES instance. The question of existence of polynomial kernels for these special cases as well as for Edge (r, ℓ)-Partization is open. Even the parameterized complexity of Edge $(2, 2)$-Partization remains open.

[1] We would like to mention that one of our results, namely, $3.3146^k n^{\mathcal{O}(1)}$ time FPT algorithm for Vertex $(2, 2)$-Partization (and hence for Vertex $(1, 2)$-Partization and Vertex $(2, 1)$-Partization) were independently and simultaneously obtained by Baste et al. (`http://arxiv.org/abs/1504.05515`).

Our methods. Most of the FPT algorithms are based on the iterative compression technique and use an algorithm for either OCT or EOCT as a subroutine. One of the algorithms also uses methods developed in [23]. To arrive at the approximation algorithm, we needed to take a detour. We start by looking at a slightly larger class of graphs called (r,ℓ)-*split graphs*. A graph G is an (r,ℓ)-split graph if its vertex set can be partitioned into V_1 and V_2 such that the size of a largest clique in $G[V_1]$ is bounded by r and the size of the largest independent set in $G[V_2]$ is bounded by ℓ. Such a bipartition for the graph G is called as (r,ℓ)-split partition. The notion of (r,ℓ)-split graphs was introduced in [15]. For any fixed r and ℓ, there is a finite forbidden set $\mathbb{F}_{r,\ell}$ for (r,ℓ)-split graphs [15]. That is, a graph G is a (r,ℓ)-split graph if and only if G does not contain any graph $H \in \mathbb{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $f(r,\ell)$, f being a function given in [15]. Since the class (r,ℓ)-graphs is a sub class of (r,ℓ)-split graphs, each graph in $\mathbb{F}_{r,\ell}$ will not appear as an induced subgraph in any (r,ℓ)-graph. For our approximation algorithm we first make the given graph (r,ℓ)-split graph by removing the induced subgraphs that are isomorphic to some graph in $\mathbb{F}_{r,\ell}$. Once we have (r,ℓ)-split graph, we generate a (r,ℓ)-split partition (V_1,V_2) of G. Then we observe that for $r,\ell \in \{1,2\}$ the problem reduces to finding an approximate solution to Odd Cycle Transversal in $G[V_1]$ and $\overline{G}[V_2]$. Finally, we use the known $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for Odd Cycle Transversal [1] to obtain a $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for our problems. The Turing kernel for Vertex (r,ℓ)-Partization, when $r,\ell \in \{1,2\}$, uses the approximation algorithm and depends on the randomized kernelization algorithm for Odd Cycle Transversal [19].

2 Preliminaries

We use standard notations from graph theory([8]) throughout this paper. The vertex set and edge set of a graph G are denoted as $V(G)$ and $E(G)$ respectively. The complement of the graph G is denoted by $\overline{G}$ and has $V(G')=V(G)$ and $\binom{V(G)}{2} - E(G)$ as its edge set. Here, $\binom{V(G)}{2}$ denotes the family of two sized subsets of $V(G)$. The neighbourhood of a vertex v is represented as $N_G(v)$, or, when the context of the graph is clear, simply as $N(v)$. An induced subgraph of G on the vertex set $V' \subseteq V$ is written as $G[V']$. An induced subgraph of G on the edge set $E' \subseteq E$ is written as $G[E']$. For a vertex subset $V' \subseteq V$, $G[V-V']$ is also denoted as $G-V'$. Similarly, for an edge set $E' \subseteq E$, $G-E'$ denotes the subgraph $G'=(V,E\setminus E')$.

The Ramsey number for a given pair of positive integers (a,b) is the minimum number such that any graph with the Ramsey number of vertices either has an independent set of size a or a clique of size b. The Ramsey number for (a,b) is denoted by $R(a,b)$.

We have already seen what (r,ℓ)-graphs are. Below, is a formal definition of the graph class as well as some related definitions.

▶ **Definition 1.** (r,ℓ)-graph A graph G is an (r,ℓ)-graph if its vertex set can be partitioned into r independent sets and ℓ cliques. We call such a partition of $V(G)$ an (r,ℓ)-partition. An IC-partition, of an (r,ℓ)-graph G, is a partition (V_1,V_2) of $V(G)$ such that $G[V_1]$ can be partitioned into r independent sets and $G[V_2]$ can be partitioned into ℓ cliques.

For fixed $r,\ell \geq 0$, the class of (r,ℓ)-graphs is closed under induced subgraphs. The following observation is useful in the understanding of the algorithms presented in the paper

▶ **Observation 2 (⋆).** [2] *Let $P=(P_I,P_C)$ and $P'=(P'_I,P'_C)$ be two IC-partitions of an (r,ℓ)-graph G. Then $|P_I \cap P'_C| \leq r\ell$ and $|P'_I \cap P_C| \leq r\ell$.*

[2] Proofs of results marked with ⋆ can be found in the full version.

3 Vertex Deletion to (r, ℓ)-graphs

In this section we first show that Vertex $(2,2)$-Partization is in FPT, using iterative compression. Then we explain how to reduce Vertex $(2,1)$-Partization and Vertex $(1,2)$-Partization to Vertex $(2,2)$-Partization. Our algorithm for Vertex $(2,2)$-Partization combines the iterative compression technique with a polynomial bound on the number of IC-partitions of a $(2,2)$-graph. The following Lemma tells about an algorithm to recognize whether a graph is a $(2,2)$-graph and also about an algorithm to compute all such IC-partitions. These results were shown in several papers [2, 9].

▶ **Lemma 3.** *Given a graph G on n vertices and m edges we can recognize whether G is a $(2,2)$-graph in $\mathcal{O}((n+m)^2)$ time. Also, a $(2,2)$-graph can have at most n^8 IC-partitions and all the IC-partitions can be enumerated in $\mathcal{O}(n^8)$ time.*

For a graph G, we say $S \subseteq V(G)$ is a $(2,2)$-vertex deletion set, if $G - S$ is a $(2,2)$-graph. Now we describe the iterative compression technique and its application to the Vertex $(2,2)$-Partization problem.

Iterative Compression for Vertex $(2,2)$-Partizipation. Let (G,k) be an input instance of Vertex $(2,2)$-Partization and let $V(G) = \{v_1, \ldots, v_n\}$. We define, for every $1 \leq i \leq |V(G)|$, the vertex set $V_i = \{v_1, \ldots, v_i\}$. Denote $G[V_i]$ as G_i. We iterate through the instances (G_i, k) starting from $i = k+5$. Given the i^{th} instances and a known $(2,2)$-vertex deletion set S'_i of size at most $k+1$, our objective is to obtain a $(2,2)$-vertex deletion set S_i of size at most k. The formal definition of this compression problem is as follows.

Vertex $(2,2)$-Partization Compression	**Parameter:** k
Input: A graph G and a $k+1$ sized vertex subset $S' \subseteq V(G)$ such that $G - S'$ is a $(2,2)$-graph	
Output: A vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is a $(2,2)$-graph	

We reduce the Vertex $(2,2)$-Partization problem to $n-k-4$ instances of the Vertex $(2,2)$-Partization Compression problem in the following manner. When $i = k+5$, the set V_{k+1} is a $(2,2)$-vertex deletion set of size at most $k+1$ for G_{k+5}. Let $I_i = (G_i, S'_i, k)$ be the i^{th} instance of Vertex $(2,2)$-Partization Compression. If S_{i-1} is a k-sized solution for I_i, then $S_{i-1} \cup \{v_i\}$ is a $(k+1)$-sized $(2,2)$-vertex deletion set for G_i. Hence, we start the iteration with the instance $I_{k+5} = (G_{k+5}, V_{k+1}, k)$ and try to obtain a $(2,2)$-vertex deletion set of size at most k. If such a solution S_{k+5} exists, we set $S'_{k+5} = S_{k+5} \cup \{v_{k+6}\}$ and ask of a k-sized solution for the instance I_{k+6}, and so on. If, during any iteration, the corresponding instance does not have a $(2,2)$-vertex deletion set of size at most k, it implies that the original instance (G,k) is a NO instance for Vertex $(2,2)$-Partization. If the input instance (G,k) is a YES instance, then S_n is a k-sized $(2,2)$-vertex deletion set for G, where $n = |V(G)|$. Since there are at most n iterations, the total time taken by the algorithm to solve Vertex $(2,2)$-Partization is at most n times the time taken to solve Vertex $(2,2)$-Partization Compression. The above explained template for doing iterative compression will be used for approximation algorithms as well as for parameterized algorithms for edge versions of these problems.

The following Lemma shows that Vertex $(2,2)$-Partization Compression is in FPT. The arguments above imply that Vertex $(2,2)$-Partization is also in FPT.

▶ **Lemma 4 ($\star$).** Vertex $(2,2)$-Partization Compression *can be solved deterministically in time* $3.3146^k |V(G)|^{\mathcal{O}(1)}$.

Proof. (Proof sketch) We design an algorithm for Vertex $(2,2)$-Partization Compression. Let (G, S') be the instance of the problem and let (P'_I, P'_C) be an IC-partition of $G - S'$. Let S be a *hypothetical solution* of size k for the problem, which the algorithm suppose to compute. Let (P_I, P_C) be an IC-partition of $G - S$. The algorithm first guesses a partition (Y, N) of S' such that $Y = S' \cap S$ and $N = S' - S$. After this guess, the objective is to compute a set Z of size at most $k' = k - |Y|$ such that $G - (Z \cup Y)$ is a $(2,2)$-graph. Also note that since N is not part of the solution S, $G[N]$ is a $(2,2)$-graph. Consider the two IC-partitions $(P_I - (S \cup S'), P_C - (S \cup S'))$ and $(P'_I - (S \cup S'), P'_C - (S \cup S'))$ of the $(2,2)$-graph $G - (S \cup S')$. By Observation 2 we know that the cardinality of each of the set $(P_I \cap P'_C) - (S \cup S')$ and $(P_C \cap P'_I) - (S \cup S')$ are bounded by 4. So now the algorithm guesses the set $V_I = (P_I \cap P'_C) - (S \cup S')$ and $V_C = (P_C \cap P'_I) - (S \cup S')$, each of them having size at most 4. After the guess of V_I and V_C, any vertex in $P'_C - V_I$ either belongs to P_C or belongs to the hypothetical solution S. Similarly any vertex in $P'_I - V_C$ either belongs to P_I or belongs to the hypothetical solution S. By Lemma 3 we know that the number of IC-partitions of $G[N]$ is at most $\mathcal{O}(k^8)$ and these partitions can be enumerate in time $\mathcal{O}(k^8)$. The algorithm now guesses an IC-partition (N_I, N_C) of $G[N]$ such that $N_I \subseteq P_I$ and $N_C \subseteq P_C$. Now consider the partition $(A, B) = ((P'_I \cup N_I \cup V_I) - V_C, (P'_C \cup N_C \cup V_C) - V_I)$. Any vertex $v \in A$ either belongs to P_I or belongs to the hypothetical solution S and any any vertex $v \in B$ either belongs to P_C or belongs to the solution S. So the objective is to find two sets $U \subseteq A$ and $W \subseteq B$ such that $G[A - U]$ is a bipartite graph, $G[B - W]$ is the complement of a bipartite graph and $|U| + |W| \leq k'$. As a consequence, the algorithm guesses the sizes k_1 of U and k_2 of W. Then the problem reduced to finding an odd cycle transversal(OCT) of size k_1 for $G[A]$ and an OCT of size k_2 for the complement of the graph $G[B]$. Hence, our algorithm runs the current best algorithm for Odd Cycle Transversal, presented in [21] for finding an OCT U of size k_1 in $G[A]$ and for finding an OCT W of size k_2 in the complement of $G[B]$. This completes the algorithm and it leads to the mentioned running time in the lemma. The running time analysis can be found in the full version. ◀

Lemma 4 and the discussions preceding it imply the following theorem.

▶ **Theorem 5.** Vertex $(2,2)$-Partization *can be solved in time* $3.3146^k |V(G)|^{\mathcal{O}(1)}$.

Vertex $(2,1)$-Partization: The Vertex $(2,1)$-Partization problem can be reduced to Vertex $(2,2)$-Partization. Suppose we are given a graph G, where $|V(G)| = n$. We construct a graph $G' = G \uplus \hat{C}$, where $\hat{C}$ is a clique on $n+3$ new vertices. That is, G' is the disjoint union of G and $\hat{C}$. The next lemma relates the graphs G and G'.

▶ **Lemma 6** ($\star$). *For any integer $t \leq n$, (G, t) is a* YES *instance of* Vertex $(2,1)$-Partization *if and only if (G', t) is a* YES *instance of* Vertex $(2,2)$-Partization. *Here $G' = G \uplus \hat{C}$ such that $\hat{C}$ is a clique on $n+3$ new vertices that are independent from G.*

Now if we are given an instance (G, k) of Vertex $(2,1)$-Partization, Lemma 6 tells us that it is enough to solve Vertex $(2,2)$-Partization on (G', k). Notice that solving the Vertex $(1,2)$-Partization problem on an input instance (G, k) is equivalent to finding a Vertex $(1,2)$-Partization on $(\overline{G}, k)$, where $\overline{G}$ is the complement graph of G. Thus, we get the following as a corollary of Theorem 5.

▶ **Corollary 7.** Vertex $(1,2)$-Partization *and* Vertex $(2,1)$-Partization *have* FPT *algorithms that run in* $3.3146^k n^{\mathcal{O}(1)}$ *time.*

4 Approximation algorithm for Vertex (r, ℓ)-Partization

In this section we give a polynomial time approximation algorithm for Vertex $(2,2)$-Partization. That is, we design an algorithm for Vertex $(2,2)$-Partization, which takes an instance (G, k), runs in polynomial time and outputs either a solution of size $\mathcal{O}(k^{3/2})$ or concludes that (G, k) is a NO instance. Since the reduction from Vertex $(2,1)$-Partization to Vertex $(2,2)$-Partization, given in Lemma 6, is an approximation preserving reduction, we can get a similar approximate algorithm for Vertex $(2,1)$-Partization. Similarly, since Vertex $(1,2)$-Partization on a graph is equivalent to Vertex $(2,1)$-Partization in the complement graph, we can get an approximation algorithm for Vertex $(1,2)$-Partization. The approximation algorithm we discuss in this section, is useful for obtaining Turing kernels for Vertex (r, ℓ)-Partization, when $1 \leq r, \ell \leq 2$. Finally, we design a factor $\mathcal{O}(\sqrt{\log n})$ approximation algorithms for these problems.

We know that (r, ℓ)-graphs is a subclass of (r, ℓ)-split graphs (See Introduction for definition). Now we give a polynomial time algorithm which takes a graph G as input and outputs an (r, ℓ)-split partition if G is an (r, ℓ)-split graph. We design such an algorithm using iterative compression. Essentially we show that the following problem, (r, ℓ)-split partition Compression, can be solved in polynomial time.

(r, ℓ)-split partition Compression
Input: A graph G with $V(G) = V \cup \{v\}$ and an (r, ℓ)-split partition (A, B) of $G[V]$
Output: An (r, ℓ)-split partition of G, if G is an (r, ℓ)-split graph, and NO otherwise

Like in the case of the FPT algorithm for Vertex $(2,2)$-Partization given in Section 3, we can show that by running the algorithm for (r, ℓ)-split partition Compression at most $n - 2$ times we can get an algorithm which outputs an (r, ℓ)-split partition of a given (r, ℓ)-split graph. Our algorithm for (r, ℓ)-split partition Compression uses the following simple lemma.

▶ **Lemma 8** (⋆). *Let G be an (r, ℓ)-split graph. Let (A, B) and (A', B') are two (r, ℓ)-split partitions of G. Then $|A \cap B'| \leq R(\ell + 1, r + 1) - 1$ and $|A' \cap B| \leq R(\ell + 1, r + 1) - 1$, where $R(r + 1, \ell + 1)$, is the Ramsey number.*

Using Lemma 8, we show that (r, ℓ)-split partition Compression can be solved in polynomial time for any fixed constants r and ℓ.

▶ **Lemma 9** (⋆). *For any fixed constants r and ℓ, (r, ℓ)-split partition Compression can be solved in polynomial time.*

By applying Lemma 9, at most $n - 2$ times, we can get the following lemma.

▶ **Lemma 10.** *For any fixed constants r and ℓ, there is an algorithm which takes a graph G as input, runs in polynomial time, and decides whether G is an (r, ℓ)-split graph. Furthermore, if G is an (r, ℓ)-split graph then the algorithm outputs an (r, ℓ)-split partition (V_1, V_2) of G.*

We know that any (r, ℓ)-graph is also an (r, ℓ)-split graph. The following lemma gives a relation between an (r, ℓ)-split partition and an IC-partition of a (r, ℓ)-graph.

▶ **Lemma 11** (⋆). *Let G be an (r, ℓ)-graph. Let (A, B) be an IC-partition of G and (A', B') be an (r, ℓ)-split partition of G. Then $|A \cap B'| \leq r\ell$ and $|A' \cap B| \leq r\ell$.*

Before giving an approximation algorithm for Vertex (r, ℓ)-Partization, we need to mention about a polynomial time approximation algorithm for Odd Cycle Transversal and finite forbidden characterization of (r, ℓ)-graphs. Using the FPT algorithm

for OCT [19], and an $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for OCT [1], one can prove the following proposition.

▶ **Proposition 12** ([19]). *There is a polynomial time algorithm which takes a graph G and an integer k as input and outputs either an OCT of G of size at most $\mathcal{O}(k^{3/2})$ or concludes that there is no OCT of size k for G.*

For any fixed r and ℓ, there is a finite forbidden set $\mathbb{F}_{r,\ell}$ for (r,ℓ)-split graphs [15]. That is, a graph G is an (r,ℓ)-split graph if and only if G does not contain any graph $H \in \mathbb{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $f(r,\ell)$, f being a function given in [15]. Since $f(2,2)$ is a constant, it is possible to compute the forbidden set $\mathbb{F}_{r,\ell}$ in polynomial time: The forbidden graphs are of size at most $f(2,2)$. Since the class (r,ℓ)-graphs is a sub class of (r,ℓ)-split graphs, each graph in $\mathbb{F}_{r,\ell}$ will not appear as an induced subgraph in any (r,ℓ)-graph. Now we are ready to design a polynomial time approximation algorithm for Vertex $(2,2)$-Partization.

▶ **Theorem 13.** *There is an algorithm which takes a graph G and an integer k as input, runs in polynomial time and outputs either a set S of size $\mathcal{O}(k^{3/2})$ such that $G-S$ is a $(2,2)$-graph or concludes that (G,k) is a* NO *instance of* Vertex $(2,2)$-Partization.

Proof. The algorithm first finds a maximal set $\mathcal{T}$ of vertex disjoint subgraphs of G such that each subgraph in $\mathcal{T}$ is isomorphic to a graph in $\mathbb{F}_{2,2}$. If $|\mathcal{T}| > k$, then clearly (G,k) is a NO instance of Vertex $(2,2)$-Partization. So the algorithm will output NO if $|\mathcal{T}| > k$. Now consider the graph $G' = G - V(\mathcal{T})$. Here, $V(\mathcal{T})$ denotes the set of vertices appearing in graphs in $\mathcal{T}$. Since $\mathcal{T}$ is a maximal set of vertex disjoint subgraphs in G which are isomorphic to a graphs in $\mathbb{F}_{2,2}$ we have that G' is a $(2,2)$-split graph.

Now our algorithm will find a set $S \subseteq V(G')$ of size bounded by $\mathcal{O}(k^{3/2})$ such that $G'-S$ is a $(2,2)$-graph. Since G' is a subgraph of G, if (G,k) is a YES instance of Vertex $(2,2)$-Partization, then (G',k) is also a YES instance. Let S^* be an hypothetical solution of the instance (G',k) of Vertex $(2,2)$-Partization and let (A,B) be an IC-partition of $G'-S^*$. Now our algorithm applies Lemma 10 on graph G' and computes a $(2,2)$-split partition (A',B') of G' in polynomial time. By Lemma 11, we know that $|A \cap B'| \leq 4$ and $|A' \cap B| \leq 4$. So the algorithm will guess the set $U = A \cap B'$ and $W = A' \cap B$. The number of possible guesses for U and W is bounded by n^8. For the correct guess U and W, we know that $A = (A' \cup U) \setminus (W \cup S^*)$ and $B = (B' \cup W) \setminus (U \cup S^*)$. Now consider the partition (V_1, V_2) of $V(G')$, where $V_1 = (A' \cup U) \setminus W$ and $V_2 = (B' \cup W) \setminus U$. So for the correct guess U and W, we know that each vertex in V_1 either belongs to A or belongs to S^* and each vertex in V_2 either belongs to B or belongs to S^*. Now to compute a solution for (G',k), it is enough to find an OCT S_1 in $G[V_1]$ and an OCT S_2 in the complement graph of $G'[V_2]$ such that $|S_1| + |S_2| = k$. Our algorithm applies Proposition 12 on $G'[V_1]$ and on the complement graph of $G'[V_2]$. If these algorithms output an OCT S_1 and an OCT S_2 for graphs $G'[V_1]$ and $\overline{G'}[V_2]$, then $S_1 \cup S_2$ is of size bounded by $\mathcal{O}(k^{3/2})$ and $G' - (S_1 \cup S_2)$ is a $(2,2)$-graph. Since $G' = G - V(\mathcal{T})$ and $G' - (S_1 \cup S_2)$ is a $(2,2)$-graph, we have that $G - (S_1 \cup S_2 \cup V(\mathcal{T}))$ is a $(2,2)$-graph. So our algorithm will output $S_1 \cup S_2 \cup V(\mathcal{T})$ as the required output. Since $|V(\mathcal{T})| \leq k \cdot f(2,2)$, we have that $|S_1 \cup S_2 \cup V(\mathcal{T})| = \mathcal{O}(k^{3/2})$. If the algorithm mentioned in Proposition 12 returns NO for all possible guesses of U and W, then our algorithm outputs NO. It is easy to see that the number of steps in our algorithm is bounded by a polynomial in $|V(G)|$. ◀

Using the arguments of Theorem 13, we can also design an approximation algorithm for finding a minimum $(2,2)$-vertex deletion set of a graph G. Let S be an optimum $(2,2)$-vertex

deletion set and (A, B) be the corresponding IC-partition of $G' = G - S$. Let $\mathcal{T}$ be a maximal set of vertex disjoint subgraphs of G, that are each isomorphic to a graph in $\mathbb{F}_{2,2}$. The number of subgraphs in $\mathcal{T}$ is at most $|S|$ and the number of vertices involved in these forbidden subgraphs is at most $f(2,2)|S|$. The remaining graph G' is a $(2,2)$-split graph and using Lemma 10, we can find a $(2,2)$-split partition (A', B') of G'. Let $(\hat{A}, \hat{B})$ be the restriction of (A, B) to G'. As argued above, at most 4 vertices from A' could be part of $\hat{B}$. Let this set of 4 vertices be called U. The rest either belong to $\hat{A}$ or S. $U \cup (S \cap A')$ is an OCT for A', of size at most $2|S \cap A'|$. The algorithm of [1] returns an $\mathcal{O}(\sqrt{\log n})$-approximate Odd Cycle Transversal solution S_1 for $G[A']$, which has to be of size at most $2|S \cap A'| \cdot \mathcal{O}(\sqrt{\log n})$. There is a similar property on the vertices of B'. Applying the algorithm of [1], on $\overline{G'[B']}$, returns an $\mathcal{O}(\sqrt{\log n})$-approximate Odd Cycle Transversal solution S_2, which has to be of size at most $2|S \cap B'| \cdot \mathcal{O}(\sqrt{\log n})$. Thus $V(\mathcal{T}) \cup S_1 \cup S_2$ is a $(2,2)$-vertex deletion set of G, with size at most $(f(2,2) + \mathcal{O}(\sqrt{\log n})|S|$. This together with Lemma 6 and discussion after that lead to the following theorem.

▶ **Theorem 14.** Vertex $(2,1)$-Partization, Vertex $(1,2)$-Partization, *and* Vertex $(2,2)$-Partization *admit polynomial time approximation algorithms with factor* $\mathcal{O}(\sqrt{\log n})$.

5 Turing Kernels for Vertex Deletion to (r, ℓ)-graphs

In this section, we give a randomized Turing kernel for Vertex $(2,2)$-Partization (See introduction for the definition). The equivalence in Lemma 6 ensures that there is a randomized Turing kernel for Vertex $(2,1)$-Partization. Since, Vertex $(1,2)$-Partization on an instance (G,k) is equivalent to Vertex $(2,1)$-Partization on $(\overline{G},k)$, a randomized Turing kernel for Vertex $(1,2)$-Partization follows.

We have seen in Section 3 that eventually the algorithm for Vertex $(2,2)$-Partization runs two instances of OCT. In this section we explain that we can use the kernelization of OCT to get a Turing kernel for Vertex $(2,2)$-Partization. A randomized polynomial kernel for OCT was shown by Kratsch and Wahlström [18], using the concept of representative family. They showed that it is possible to find $k^{\mathcal{O}(1)}$ "relevant" vertices from the input graph which contains the optimum solution. This leads to a randomized kernel for OCT. In fact, the following lemma follows from the work of Kratsch and Wahlström. We give a proof for the lemma in the full version.

▶ **Lemma 15** ($\star$). *Let G be a graph and X be an OCT of G. There is a randomized polynomial time algorithm which computes a set $Z \subseteq V(G)$ of size $\mathcal{O}(|X|^3)$ such that for any $Y \subseteq X$, a minimum sized OCT, of $G - Y$, is fully contained in Z.*

Now we are ready to explain our Turing kernel for Vertex $(2,2)$-Partization using Lemma 15. Given an instance (G,k) of Vertex $(2,2)$-Partization, first we construct $|V(G)|^{\mathcal{O}(1)}$ many instances of a problem which is in NP and each of them have size bounded by polynomial in k. Then, by using the Cook-Levin theorem [5], we can reduce each of these intances to instances of Vertex $(2,2)$-Partization and thus arrive at a Turing kernelization for Vertex $(2,2)$-Partization. We first run the polynomial time approximation algorithm described in Theorem 13. If the approximation algorithm outputs NO, then the algorithm will output a trivial NO instance of the problem. Otherwise let X be the solution returned by the approximation algorithm on input (G,k). We know that the cardinality of X is bounded by $\mathcal{O}(k^{3/2})$. Now we fix an IC-partition (P_I, P_C) of $G - X$. Let S be a *hypothetical* solution of size at most k and (Q_I, Q_C) be an IC-partition of $G - S$. It follows from Observation 2 that $|P_I \cap Q_C| \leq 4$ and $|Q_I \cap P_C| \leq 4$. This observation leads to the following lemma.

▶ **Lemma 16.** *(G, k) is a* YES *instance of* Vertex $(2,2)$-Partization *if and only if there exist $V_C \subseteq P_I$ and $V_I \subseteq P_C$, each of cardinality at most 4 such that $X' = X \cup V_C \cup V_I$ can be partitioned into X'_I, X'_D, X'_C, with the following properties:*

1. *There is a set $Z_I \subseteq (P_I \setminus V_C) \cup X'_I$ such that $Z_I \cup X'_D \cup X'_C$ is an OCT for $G[P_I \cup X']$. In other words, Z_I is an OCT for $G[P_I \cup X'_I]$.*
2. *There is a set $Z_C \subseteq (P_C \setminus V_I) \cup X'_C$ such that $Z_C \cup X'_D \cup X'_I$ is an OCT for $\overline{G}[P_C \cup X']$. In other words, Z_C is an OCT for $\overline{G}[P_C \cup X'_C]$.*
3. *$|Z_I \cup Z_C \cup X'_D| \leq k$.*

Proof. Suppose (G, k) is a YES instance of Vertex $(2,2)$-Partization. Then there is a k-sized solution Z such that $G - Z$ is a $(2,2)$-graph. Let (Q_I, Q_C) be an IC-partition of $G - Z$. Let $V_C = P_I \cap Q_C$ and $V_I = P_C \cap Q_I$. It follows from Observation 2 that $|V_I| \leq 4$ and $|V_C| \leq 4$. Notice that any vertex in $P_I \setminus V_C$ either belongs to Q_I or to Z. Similarly, any vertex in $P_C \setminus V_I$ either belongs to Q_C or to Z. Let $X' = X \cup V_I \cup V_C$. Now we define $X'_I = X' \cap Q_I$, $X'_C = X' \cap Q_C$ and $X'_D = X' \cap Z$. Let $Z_I = Z \cap P_I$ and $Z_C = Z \cap P_C$. Note that $Z_I \cap V_C = \emptyset$ and $Z_C \cap V_I = \emptyset$. From the definition of X', V_I and V_C, it is clear that $V_I \subseteq X'_I$ and $V_C \subseteq X'_C$. Since $V_C \subseteq X'_I$ and $V_C \subseteq X'_C$, we have that $(P_I \cup X') \setminus (Z_I \cup X'_D \cup X'_C) = Q_I$. Also since, $G[Q_I]$ is a bipartite graph we have that $(Z_I \cup X'_D \cup X'_C)$ is an OCT of $G[P_I \cup X']$. By similar arguments we can show that $(Z_C \cup X'_D \cup X'_I)$ is an OCT of $\overline{G}[P_C \cup X']$. Since $Z_I \cup Z_C \cup X'_D = Z$ and $|Z| = k$, the set $Z_I \cup Z_C \cup X'_D$ satisfies condition 3 in the lemma. This completes the proof of the forward direction.

Conversely, suppose there is a $V_C \subseteq P_I$ and $V_I \subseteq P_C$, each of size at most 4 such that the $X' = X \cup V_I \cup V_C$ has a 3-partition $(X'_I \cup X'_D \cup X'_C)$ with the properties mentioned in the lemma. That is, there is an OCT Z_I for the graph $G[P_I \cup X'_I]$ and an OCT Z_C for the graph $\overline{G}[P_C \cup X'_C]$ such that $|Z_I \cup Z_C \cup X'_D| \leq k$. Then we claim that $Z = Z_I \cup Z_C \cup X'_D$ is a $(2,2)$-vertex deletion set of G. Consider the sets $Q_I = (P_I \cup X'_I) \setminus Z_I$ and $Q_C = (P_C \cup X'_C) \setminus Z_C$. By our assumption $G[Q_I]$ and $\overline{G}[Q_C]$ are bipartite graphs. Also note that $Q_I \cup Q_C \cup Z = V(G)$. Hence Z is a $(2,2)$-vertex deletion set of G and (Q_I, Q_C) is an IC-partition of $G - Z$. ◀

The Lemma 16 allows us to reduce an instance of Vertex $(2,2)$-Partization to polynomially many instances of a problem which is in NP. Consider the following problem.

Twin Odd Cycle Transversal (TOCT) **Parameter:** k
Input: Two graphs G_1 and G_2, terminals $X \subseteq V(G_1)$, $Y \subseteq V(G_2)$, a bijection Φ between X and Y, and an integer k
Question: Is there a partition of X into three parts (X_1, X_D, X_2) such that there is an OCT $Z_1 \subseteq V(G_1) \setminus (X_D \cup X_2)$ for the graph $G_1 - (X_D \cup X_2)$, an OCT $Z_2 \subseteq V(G_2) \setminus (\Phi(X_D) \cup \Phi(X_1))$ for the graph $G_2 - (\Phi(X_D) \cup \Phi(X_1))$ and $\lvert Z_1 \cup X_D \cup Z_2 \rvert \leq k$?

Clearly the problem TOCT is in NP. Because of Lemma 16, for each $V_C \subseteq P_1$ and $V_I \subseteq P_C$ of cardinality at most 4, we construct an instance of TOCT, of size bounded by a polynomial in k, using Lemma 15. After this, we fix a $V_I \subseteq P_C$ and a $V_C \subseteq P_I$, each of cardinality at most 4. Now let $X' = X \cup V_I \cup V_C$. Note that X' is a $(2,2)$-vertex deletion set of G and $(P_I \setminus V_C, P_C \setminus V_I)$ is an IC-partition of $G - X'$. The following observation is derived from the fact that $(P_I \setminus V_C, P_C \setminus V_I)$ is an IC-partition of $G - X'$ and $V_I \cup V_C \subseteq X'$.

▶ **Observation 17.** *The set X' is an OCT of $G[P_I \cup X']$ and also an OCT of $\overline{G}[P_C \cup X']$.*

For a particular choice of $V_C \subseteq P_I$ and $V_I \subseteq P_C$ of cardinality at most 4, we construct an instance of TOCT as follows. Let $X' = X \cup V_I \cup C_C$, where X is the approximate solution of size bounded by $\mathcal{O}(k^{3/2})$. Let (P_I, P_C) be an IC-partition of $G - X$. Let $G_1 = G[P_I \cup X']$ and $G_2 = \overline{G}[P_C \cup X']$. By Observation 17, X' is an OCT in graphs G_1 and G_2. Now we

apply Lemma 15 and get a set of relevant vertices $Z_1 \subseteq V(G_1)$ of size bounded by $\mathcal{O}(k^{9/2})$. Next, we construct a graph G_1^* as follows: delete all the vertices $V(G_1) \setminus (X' \cup Z_1)$ from G_1. Add two length (three length) path between two vertices in $V(G_1^*)$, if there is an even length (odd length) path between the corresponding vertices in G_1 using only vertices from $V(G) \setminus (X' \cup Z_1)$. Similarly, we construct a graph G_2^* from G_2. Now we output $H = (G_1, G_2, X', X', k)$ as the reduced instance of TOCT, with the bijection between X' and X' be the natural identity map. Since there are $\mathcal{O}(n^4)$ choices for selecting V_C and V_I, our algorithm will output instances $H_1, H_2, \ldots H_t$ where $t = \mathcal{O}(n^4)$ and the size of each H_i is bounded by $\mathcal{O}(k^9)$.

Using Lemmata 15 and 16 we can prove that in fact the above Turing reduction is correct.

▶ **Lemma 18** (⋆). *(G, k) is a* YES *instance of* Vertex $(2,2)$-Partization *if and only if there exists i such that H_i is a* YES *instance of* TOCT.

The problem TOCT is in NP and Vertex $(2,1)$-Partization is NP-complete. Therefore, by Cook-Levin theorem each instance H_i of TOCT can be reduced to an an instance of Vertex $(2,2)$-Partization in polynomial time. Also note that size of each instance H_i is bounded by $\mathcal{O}(k^9)$. Thus we have the following theorem.

▶ **Theorem 19.** *There exists a randomized polynomial Turing kernel for* Vertex $(2,2)$-Partization.

Since there is parameter preserving reduction from Vertex $(2,1)$-Partization and Vertex $(1,2)$-Partization to Vertex $(2,2)$-Partization, the following corollary is derived from Theorem 19.

▶ **Corollary 20.** *There exists a randomized polynomial Turing kernel for* Vertex $(2,1)$-Partization *and* Vertex $(1,2)$-Partization.

6 Edge Deletion to (r, ℓ)-graphs

In this section we show that Edge $(2,1)$-Partization and Edge $(1,2)$-Partization are in FPT.

6.1 Edge $(2,1)$-Partization

In this subsection we show that Edge $(2,1)$-Partization is in FPT, using iterative compression. For Edge $(2,1)$-Partization, the corresponding compression problem is defined as follows.

Edge $(2,1)$-Partization Compression	**Parameter:** k
Input: A graph G with $V(G) = V \cup \{v\}$, an integer k and an edge set $S' \subseteq E(G - \{v\})$, of size at most k, such that $G[V] - S'$ is a $(2,1)$-graph	
Output: A subset $S \subseteq E$ of size at most k such that $G - S$ is a $(2,1)$-graph	

Similar to Vertex $(2,2)$-Partization, we can show that Edge $(2,1)$-Partization can be solved, by running Edge $(2,1)$-Partization Compression at most $|V(G)|$ times, for an input instance (G, k). The following lemma is useful for our purpose.

▶ **Lemma 21** (⋆). *Let G be a graph on n vertices, $v \in V(G)$ and $|E(G - \{v\})| \leq k$. Then the number of cliques in G is bounded by $2^{\mathcal{O}(\sqrt{k})}n$ and these cliques can be enumerated in time $2^{\mathcal{O}(\sqrt{k})}n$.*

Next we show that Edge $(2,1)$-Partization Compression is in FPT.

▶ **Lemma 22** ($\star$). *Edge $(2,1)$-Partization Compression is solved in time $2^{k+o(k)}n^{\mathcal{O}(1)}$.*

Thus, by using Lemma 22, we prove the following theorem.

▶ **Theorem 23.** *Edge $(2,1)$-Partization can be solved in time $2^{k+o(k)}n^{\mathcal{O}(1)}$.*

6.2 Edge $(1,2)$-Partization

In this subsection we show that Edge $(1,2)$-Partization is in FPT. Again we use the iterative compression technique to solve the problem. For our algorithm, we need an algorithm for a version of Odd Cycle Transversal. Let $\mathbb{G}$ be an hereditary graph class (hereditary means that if $G \in \mathbb{G}$, then every induced subgraph of G is in $\mathbb{G}$ as well) and $\mathbb{G}$ is decidable. Then the problem $\mathbb{G}$-Weighted Bipartition is defined as follows.

$\mathbb{G}$-Weighted Bipartition	**Parameter:** $k+W$
Input: A graph G, $w : V(G) \to \mathbb{N}^+$ and integers k and W	
Output: An OCT O of G, of size at most k such that $w(O) \leq W$ and $G[O] \in \mathbb{G}$	

Marx et al. [23] showed that the unweighted version of the problem, $\mathbb{G}$-Bipartition can be solved in FPT time. The proof by Marx et al., constructs an "equivalent graph" with treewidth bounded by a function of k. The problem is then solved in the equivalent graph, using Courcelle's theorem [6] by expressing the problem as an MSO predicate. Since we can express whether the weight of a subset of vertices is at most W using an MSO predicate of length bounded by a function of W, the following theorem follows from the results of Marx et al. [23].

▶ **Theorem 24.** *If $\mathbb{G}$ is hereditary and decidable, then $\mathbb{G}$-Weighted Bipartition is in* FPT.

Now we are ready to define compression version of the problem Edge $(1,2)$-Partization and prove that it is in FPT, which in turn will imply that non-compression version of the problem is in FPT.

Edge $(1,2)$-Partization Compression	**Parameter:** k
Input: A Graph G with $V(G) = V \cup \{v\}$, an integer k and an edge set $S' \subseteq E(G-v)$, of size at most k, such that $G[V] - S'$ is a $(1,2)$-graph	
Output: A subset $S \subseteq E$ of size at most k such that $G - S$ is a $(1,2)$-graph	

▶ **Lemma 25** ($\star$). *Edge $(1,2)$-Partization Compression is in* FPT.

Thus by using Lemma 25, we get the following theorem.

▶ **Theorem 26.** *Edge $(1,2)$-Partization is in* FPT.

7 Conclusion

In this paper we explored parameterized complexity of a family of partition problems, namely Vertex (r,ℓ)-Partization and Edge (r,ℓ)-Partization. Whether there exists a polynomial kernel for these problems remains an interesting open problem. Also, the parameterized complexity of Edge $(2,2)$-Partization remains unresolved.

References

1 Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min uncut, min 2cnf deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

2 Andreas Brandstädt, Van Bang Le, and Thomas Szymczak. The complexity of some problems related to graph 3-colorability, 1998.

3 Chandra Chekuri, editor. *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. SIAM, 2014.

4 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

5 Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, New York, NY, USA, 1971. ACM.

6 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

7 Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Inf. Process. Lett.*, 113(5-6):179–182, 2013.

8 R. Diestel. *Graph Theory*. Springer, Berlin, second ed., electronic edition, February 2000.

9 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *STOC*, 16:464–472, 2003.

10 Tomás Feder, Pavol Hell, and Shekoofeh Nekooei Rizi. Partitioning chordal graphs. *Electronic Notes in Discrete Mathematics*, 38:325–330, 2011.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

13 Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015.

14 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

15 András Gyárfás. Generalized split graphs and ramsey numbers. *J. Comb. Theory, Ser. A*, 81(2):255–261, 1998.

16 Falk Hüffner. Algorithm engineering for optimal graph bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009.

17 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In Chekuri [3], pages 1749–1761.

18 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS*, pages 450–459, 2012.

19 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms*, 10(4):20, 2014.

20 R. Krithika and N. S. Narayanaswamy. Parameterized algorithms for (r, l)-partization. *J. Graph Algorithms Appl.*, 17(2):129–146, 2013.

21 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15, 2014.

22 Daniel Lokshtanov, Saket Saurabh, and Somnath Sikdar. Simpler parameterized algorithm for oct. In Jirí Fiala, Jan Kratochvíl, and Mirka Miller, editors, *IWOCA*, volume 5874 of *Lecture Notes in Computer Science*, pages 380–384. Springer, 2009.

23 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013.

24 M. S. Ramanujan and Saket Saurabh. Linear time parameterized algorithms via skew-symmetric multicuts. In Chekuri [3], pages 1739–1748.

25 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

Finding Even Subgraphs Even Faster*

Prachi Goyal[1], Pranabendu Misra[2], Fahad Panolan[2], Geevarghese Philip[4], and Saket Saurabh[2,3]

1 **Indian Institute of Science, Bangalore, India**
prachi.goyal@csa.iisc.ernet.in
2 **Institute of Mathematical Sciences, Chennai, India**
{pranabendu,fahad,saket}@imsc.res.in
3 **University of Bergen, Norway**
4 **Chennai Mathematical Institute, India**
gphilip@cmi.ac.in

Abstract

Problems of the following kind have been the focus of much recent research in the realm of parameterized complexity: *Given an input graph (digraph) on n vertices and a positive integer parameter k, find if there exist k edges (arcs) whose deletion results in a graph that satisfies some specified parity constraints.* In particular, when the objective is to obtain a connected graph in which all the vertices have even degrees – where the resulting graph is *Eulerian* the problem is called Undirected Eulerian Edge Deletion. The corresponding problem in digraphs where the resulting graph should be strongly connected and every vertex should have the same in-degree as its out-degree is called Directed Eulerian Edge Deletion. Cygan et al. [*Algorithmica, 2014*] showed that these problems are fixed parameter tractable (FPT), and gave algorithms with the running time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. They also asked, as an open problem, whether there exist FPT algorithms which solve these problems in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. It was also posed as an open problem at the School on Parameterized Algorithms and Complexity 2014, Będlewo, Poland. In this paper we answer their question in the affirmative: using the technique of computing *representative families of co-graphic matroids* we design algorithms which solve these problems in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. The crucial insight we bring to these problems is to view the solution as an independent set of a co-graphic matroid. We believe that this view-point/approach will be useful in other problems where one of the constraints that need to be satisfied is that of connectivity.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Eulerian Edge Deletion, FPT, Representative Family

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.434

1 Introduction

Many well-studied algorithmic problems on graphs can be phrased in the following way: Let $\mathcal{F}$ be a family of graphs or digraphs. Given as input a graph (digraph) G and a positive integer k, can we delete k vertices (or edges or arcs) from G such that the resulting graph (digraph) belongs to the class $\mathcal{F}$? Recent research in parameterized algorithms has focused on problems of this kind where the class $\mathcal{F}$ consists of all graphs/digraphs whose vertices

* Supported by the European Research Council (ERC) via grant PARAPPROX, reference 306992; and by the Department of Science and Technology (DST), Government of India, the German Federal Ministry of Education and Research (BMBF), and the Max Planck Society (MPG), via the Indo-German Max Planck Center for Computer Science(IMPECS)

satisfy certain *parity* constraints [4, 9, 2, 8, 5]. In this paper we obtain significantly faster parameterized algorithms for two such problems, improving the previous best bounds due to Cygan et al. [4]. We also settle the parameterized complexity of a third problem, disproving a conjecture of Cai and Yang [2] and solving an open problem posed by Fomin and Golovach [9]. We obtain our results using recently-developed techniques for the efficient computation of representative sets of matroids.

An undirected graph G is *even* (respectively, *odd*) if every vertex of G has even (resp. odd) degree. A directed graph D is *balanced* if the in-degree of each vertex of D is equal to its out-degree. An undirected graph is *Eulerian* if it is connected and even; and a directed graph is *Eulerian* if it is strongly connected and balanced. Cai and Yang [2] initiated the systematic study of parameterized Eulerian subgraph problems. In this work we take up the following edge-deletion problems of this kind:

Undirected Eulerian Edge Deletion	**Parameter:** k
Input: A connected undirected graph G and an integer k.	
Question: Does there exist a set S of at most k edges in G such that $G \setminus S$ is Eulerian?	

Undirected Connected Odd Edge Deletion	**Parameter:** k
Input: A connected undirected graph G and an integer k.	
Question: Does there exist a set S of at most k edges in G such that $G \setminus S$ is odd and connected?	

Directed Eulerian Edge Deletion	**Parameter:** k
Input: A strongly connected directed graph D and an integer k.	
Question: Does there exist a set S of at most k arcs in D such that $D \setminus S$ is Eulerian?	

Our algorithms for these problems also find such a set S of edges/arcs when it exists; so we slightly abuse the notation and refer to S as a *solution* to the problem in each case.

Previous Work. Cai and Yang [2] listed sixteen odd/even undirected subgraph problems in their pioneering paper, and settled the parameterized complexity of all but four. The first two problems above are among these four; Cai and Yang conjectured that these are both W[1]-hard, and so are unlikely to have fixed-parameter tractable (FPT) algorithms: those with running times of the form $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f where n is the number of vertices in the input graph. Cygan et al. [4] disproved this conjecture for the first problem: they used a novel and non-trivial application of the colour-coding technique to solve both Undirected Eulerian Edge Deletion and Directed Eulerian Edge Deletion in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. They also posed as open the question whether there exist $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time algorithms for these two problems. It was also posed as an open problem at the School on Parameterized Algorithms and Complexity 2014, Będlewo, Poland [3]. Fomin and Golovach [9] settled the parameterized complexity of the other two problems – not defined here – left open by Cai and Yang, but left the status of Undirected Connected Odd Edge Deletion open.

Our Results and Methods. We devise deterministic algorithms of running time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ for all the three problems defined above. This answers the question of Cygan et al. [4] in the affirmative, solves the problem posed by Fomin and Golovach, and disproves the conjecture of Cai and Yang for Undirected Connected Odd Edge Deletion.

▶ **Theorem 1.1.** Undirected Eulerian Edge Deletion, Undirected Connected Odd Edge Deletion, *and* Directed Eulerian Edge Deletion *can all be solved in*

time $\mathcal{O}(2^{(2+\omega)k} \cdot n^2 m^3 k^6) + m^{\mathcal{O}(1)}$ *where* $n = |V(G)|$, $m = |E(G)|$ *and* ω *is the exponent of matrix multiplication.*

Our main conceptual contribution is *to view the solution as an independent set of a co-graphic matroid*, which we believe will be useful in other problems where one of the constraints that need to be satisfied is that of connectivity.

We now give a high-level overview of our algorithms. Given a subset of vertices T of a graph G, a *T-join* of G is a set $S \subseteq E(G)$ of edges such that T is exactly the set of odd degree vertices in the subgraph $H = (V(G), S)$. Observe that T-joins exist only for even-sized vertex subsets T. The following problem is long known to be solvable in polynomial time [7].

Min T-Join
Input: An undirected graph G and a set of terminals $T \subseteq V(G)$.
Question: Find a T-join of G of the smallest size.

Consider the two problems we get when we remove the connectivity (resp. strong connectivity) requirement on the graph $G \setminus S$ from Undirected Eulerian Edge Deletion and Directed Eulerian Edge Deletion; we call these problems Undirected Even Edge Deletion and Directed Balanced Edge Deletion, respectively. Cygan et al. show that Undirected Even Edge Deletion can be reduced to Min T-Join, and Directed Balanced Edge Deletion to a minimum cost flow problem with unit costs, both in polynomial time [4]. Thus it is not the local requirement of even degrees which makes these problems hard, but the simultaneous global requirement of (strong) connectivity.

To handle this situation we turn to a *matroid* which correctly captures the connectivity requirement. Let $\mathcal{I}$ be the family of all subsets $X \subseteq E(G)$ of the edge set of a graph G such that the subgraph $(V(G), E(G) \setminus X)$ is connected. Then the pair $(E(G), \mathcal{I})$ forms a linear matroid called the co-graphic matroid of G (See Section 2 for definitions). Let T be the set of odd-degree vertices of the input graph G. Observe that for Undirected Eulerian Edge Deletion, the solution S we are after is *both* a T-join *and* an independent set of the co-graphic matroid of G. We exploit this property of S to design a dynamic programming algorithm which finds S by computing "representative sub-families" [10, 12, 14, 15] of certain families of edge subsets in the context of the co-graphic matroid of G. We give simple characterizations of solutions which allow us to do dynamic programming, where at every step we only need to keep a representative family of the family of partial solutions where each partial solution is an independent set of the corresponding co-graphic matroid. To find the desired representative family of partial solutions we use the algorithm by Lokshtanov et al. [13]. Our methods also imply that Undirected Connected Odd Edge Deletion admits an algorithm with running time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.

2 Preliminaries

Throughout the paper we use ω to denote the exponent in the running time of matrix multiplication, the current best known bound for which is $\omega < 2.373$ [17].

Graphs and Directed Graphs. We use "graph" to denote simple graphs without self-loops, directions, or labels, and "directed graph" or "digraph" for simple directed graphs without self-loops or labels. We use standard terminology from the book of Diestel [6] for those graph-related terms which we do not explicitly define. In general we use G to denote a graph and D to denote a digraph. We use $V(G)$ and $E(G)$, respectively, to denote the vertex and edge sets of a graph G, and $V(D)$ and $A(D)$, respectively, to denote the vertex and arc

sets of a digraph D. For an edge set $E' \subseteq E(G)$, we use (i) $V(E')$ to denote the set of *end vertices* of the edges in E', (ii) $G \setminus E'$ to denote the subgraph $G' = (V(G), E(G) \setminus E')$ of G, and (iii) $G(E')$ to denote the subgraph $(V(E'), E')$ of G. The terms $V(A')$, $D \setminus A'$, and $D(A')$ are defined analogously for an arc subset $A' \subseteq A(D)$.

If P is a path from vertex u to vertex v in graph G (or in digraph D) then we say that (i) P *connects* u and v, (ii) u, v are, respectively, the *initial vertex* and the *final* vertex of P, and (iii) u, v are the *end* vertices of path P. Let $P_1 = x_1x_2 \dots x_r$ and $P_2 = y_1y_2 \dots y_s$ be two *edge-disjoint* paths in graph G. If $x_r = y_1$ and $V(P_1) \cap V(P_2) = \{x_r\}$, then we use P_1P_2 to denote the path $x_1x_2 \dots x_ry_2 \dots y_s$. A *path system* $\mathcal{P}$ in graph G (resp., digraph D) is an ordered collection of paths in G (resp. in D), and it is *edge-disjoint* if no two paths in the system share an edge. We use $V(\mathcal{P})$ and $E(\mathcal{P})$ ($A(\mathcal{P})$ for a path system in digraph) for the set of vertices and edges, respectively, in a path system $\mathcal{P}$. We say that a path system $\mathcal{P} = \{P_1, \dots, P_r\}$ *ends* at a vertex u if the path P_r ends at u, and u is called the *final vertex* of $\mathcal{P}$. We use $V^e(\mathcal{P})$ to denote the set of end vertices of paths in a path system $\mathcal{P}$. For a path system $\mathcal{P}$ in a digraph D, we use $V^i(\mathcal{P})$ and $V^f(\mathcal{P})$, respectively, to denote the set of initial vertices and the set of final vertices, respectively, of paths in $\mathcal{P}$. For a path system $\mathcal{P} = \{P_1, \dots, P_r\}$ and an edge/arc (u, v), we define $\mathcal{P} \circ (u, v)$ as follows.

$$\mathcal{P} \circ (u,v) = \begin{cases} \{P_1, \dots, P_rv\} & \text{if } u \text{ is the final vertex of } P_r \text{ and } v \notin V(P_r) \\ \{P_1, \dots, P_r, uv\} & \text{if } u \text{ is not the final vertex of } P_r \end{cases}$$

A directed graph D is *strongly connected* if for any two vertices u and v of D, there is a directed path from u to v and a directed path from v to u in D. A digraph D is *weakly connected* if the underlying undirected graph is connected. The *in-neighborhood* of a vertex v in D is the set $N^-_D(v) = \{u \,|\, (u, v) \in A(D)\}$, and the *in-degree* of v in D is $d^-_D(v) = |N^-_D(v)|$. The *out-neighborhood* of v is the set $N^+_D(v) = \{w \,|\, (v, w) \in A(D)\}$, and its *out-degree* is $d^+_D(v) = |N^+_D(v)|$.

Co-Graphic Matroids. The *co-graphic matroid* of a connected graph G is defined as $M = (E(G), \mathcal{I})$ where $\mathcal{I} = \{S \subseteq E(G) \mid (G \setminus S) \text{ is connected}\}$. It is a linear matroid and, given a graph G, a representation of the co-graphic matroid of G over the finite field $\mathbb{F}_2$ can be found in polynomial time [14, 16]. The rank of the cographic matroid of a connected graph G is $(|E(G)| - |V(G)| + 1)$. We use M_G to denote the co-graphic matroid of a graph G. For a directed graph D we use M_D to denote the co-graphic matroid of the underlying undirected graph of D.

Let $\mathcal{A}$ be a family of path systems in a graph G. Let $e = (u, v)$ be an edge in G (or an arc in D), and let $M = (E, \mathcal{I})$ be the co-graphic matroid of graph G (or of digraph D). We use $\mathcal{A} \bullet \{e\}$ to denote the family of path systems

$$\mathcal{A} \bullet \{e\} = \{\mathcal{P}' = \mathcal{P} \circ e \mid \mathcal{P} \in \mathcal{A}, e \notin E(\mathcal{P}), E(\mathcal{P}') \in \mathcal{I}\}.$$

Representative Families of Matroids. The notion of representative families of matroids and their fast computation play key roles in our algorithms.

▶ **Definition 2.1.** **[10, 14]** Given a matroid $M = (E, \mathcal{I})$, a family $\mathcal{S}$ of subsets of E, and a non-negative integer q, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is *q-representative* for $\mathcal{S}$ if the following holds. For every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$.

In other words, if some independent set X in $\mathcal{S}$ can be extended to a larger independent set by a set Y of at most q new elements, then there is a set $\widehat{X}$ in $\widehat{\mathcal{S}}$ that can be extended by the *same* set Y. If $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q-representative for $\mathcal{S}$ we write $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$.

In this paper we are interested in linear matroids and in representative families derived from them. The following theorem states the key algorithmic result which we use for the computation of representative families of linear matroids.

▶ **Theorem 2.2** ([13]). *Let $M = (E, \mathcal{I})$ be a linear matroid of rank n and let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a family of independent sets, each of size b. Let A be an $n \times |E|$ matrix representing M over a field $\mathbb{F}$, where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or $\mathbb{F}$ is $\mathbb{Q}$. Then there is deterministic algorithm which computes a representative set $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ of size at most $nb\binom{b+q}{b}$, using $\mathcal{O}\Big(\binom{b+q}{b}tb^3n^2 + t\binom{b+q}{b}^{\omega-1}(bn)^{\omega-1}\Big) + (n+|E|)^{\mathcal{O}(1)}$ operations over the field $\mathbb{F}$.*

3 Undirected Eulerian Edge Deletion

In this section we describe our $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time algorithm for Undirected Eulerian Edge Deletion. Let (G, k) be an instance of the problem. Cygan et al. [4] observed the following characterization.

▶ **Observation 3.1.** *A set $S \subseteq E(G)\,; |S| \leq k$ of edges of a graph G is a solution to the instance (G, k) of* Undirected Eulerian Edge Deletion *if and only if it satisfies the following conditions:*
(a) *$G \setminus S$ is a connected graph; and,*
(b) *S is a T-join where T is the set of all odd degree vertices in G.*

For a designated set $T \subseteq V(G)$ of *terminal* vertices of graph G, we call a set $S \subseteq E(G)$ a *co-connected T-join* of graph G if (i) $G \setminus S$ is connected and (ii) S is a T-join. From Observation 3.1 we get that the Undirected Eulerian Edge Deletion problem is equivalent to checking whether the given graph G has a co-connected T-join of size at most k, where T is the set of all *odd-degree* vertices in G. We present an algorithm which finds a co-connected T-join for an *arbitrary* (even-sized) set of terminals T within the claimed time-bound. That is, we solve the following more general problem:

Co-Connected T-Join	**Parameter:** k
Input: A connected graph G, an even-sized subset $T \subseteq V(G)$ and an integer k.	
Question: Does there exist a co-connected T-join of G of size at most k?	

We design a dynamic programming algorithm for this problem where the partial solutions which we store satisfy the first property of co-connected T-join and "almost satisfy" the second property. To limit the number of partial solutions which we need to store, we compute and store instead, at each step, a *representative family* of the partial solutions in the corresponding co-graphic matroid. We start with the following characterization of the T-joins of a graph G.

▶ **Proposition 3.2** ([11, Proposition 1.1]). *Let T be an even-sized subset of vertices of a graph G, and let $\ell = \frac{|T|}{2}$. A subset S of edges of G is a T-join of G if and only if S can be expressed as a union of the edge sets of (i) ℓ paths which connect disjoint pairs of vertices in T, and (ii) zero or more cycles, where the paths and cycles are all pairwise edge-disjoint.*

This proposition yields the following useful property of *inclusion-minimal* co-connected T-joins (*minimal* co-connected T-joins for short) of a graph G.

▶ **Lemma 3.3 (♣).** [1] *Let T be an even-sized subset of vertices of a graph G, and let $\ell = \frac{|T|}{2}$. Let S be a minimal co-connected T-join of G. Then (i) the subgraph $G(S)$ is a forest, and (ii) the set S is a union of the edge-sets of ℓ pairwise edge disjoint paths which connect disjoint pairs of vertices in T.*

Note that the set of paths described in Lemma 3.3 are just pairwise *edge-disjoint*. *Vertices* (including terminals) may appear in more than one path as *internal* vertices. A partial converse of the above lemma follows directly from Proposition 3.2.

▶ **Lemma 3.4 (♣).** *Let T be an even-sized subset of vertices of a graph G, and let $\ell = \frac{|T|}{2}$. Let a subset $S \subseteq E(G)$ of edges of G be such that (i) $G \setminus S$ is connected, and (ii) S is a union of the edge-sets of ℓ pairwise edge-disjoint paths which connect disjoint pairs of vertices in T. Then S is a co-connected T-join.*

An immediate corollary of Lemma 3.3 is that for any set $T \subseteq V(G)$, *any* T-join of the graph G has at least $|T|/2$ edges. Hence if $|T| > 2k$ then we can directly return No as the answer for Co-Connected T-Join. So from now on we assume that $|T| \leq 2k$. From Lemmas 3.3 and 3.4 we get that to solve Co-Connected T-Join it is enough to check for the existence of a pairwise edge-disjoint collection of paths $\mathcal{P} = \{P_1, \ldots, P_{\frac{|T|}{2}}\}$ such that (i) the subgraph $(G \setminus E(\mathcal{P}))$ is connected, (ii) $|E(\mathcal{P})| \leq k$, and (iii) the paths in $\mathcal{P}$ connect disjoint pairs of terminals in T. We use dynamic programming to find such a path system.

We first state some notation which we need to describe the dynamic programming table. We use $\mathcal{Q}$ to denote the set of *all* path systems in G which satisfy the above conditions. For $1 \leq i \leq k$ we use $\mathcal{Q}^{(i)}$ to denote the set of all *potential partial* solutions of *size* i : Each $\mathcal{Q}^{(i)}$ is a collection of path systems $\mathcal{Q}^{(i)} = \{\mathcal{P}_1^{(i)}, \ldots, \mathcal{P}_t^{(i)}\}$ where each path system $\mathcal{P}_s^{(i)} = \{P_1, \ldots, P_r\} \in \mathcal{Q}^{(i)}$ has the following properties:

- **(i)** The paths $P_1, \ldots, P_r$ are pairwise edge-disjoint.
- **(ii)** The end-vertices of the paths $P_1, \ldots, P_r$ are all terminals and are pairwise disjoint, *with one possible exception*. One end-vertex (the *final* vertex) of the path P_r may be a non-terminal, or a terminal which appears as an end-vertex of another path as well.
- **(iii)** $|E(\mathcal{P}_s^{(i)})| = i$, and the subgraph $G \setminus E(\mathcal{P}_s^{(i)})$ is connected.

Note that the only ways in which a partial solution $\mathcal{P}_s^{(i)}$ may violate one of the conditions in Lemma 3.4 are: (i) it may contain strictly less than $\frac{|T|}{2}$ paths, and/or (ii) there may be a path P_r (and only one such), which has *one* end-vertex v_r which is a non-terminal or is a terminal which is an end-vertex of another path as well. For a path system $\mathcal{P} = \{P_1, \ldots, P_r\}$ and $u \in V(G) \cup \{\epsilon\}$, we use $W(\mathcal{P}, u)$ to denote the following set.

$$W(\mathcal{P}, u) = \begin{cases} V^e(\mathcal{P}) & \text{if } u = \epsilon \\ (V^e(\mathcal{P} \setminus \{P_r\})) \cup \{v \mid v \text{ is the initial vertex of } P_r\} & \text{if } u \neq \epsilon \end{cases}$$

Finally, for each $1 \leq i \leq k$, $T' \subseteq T$, and $v \in (V(G) \cup \{\epsilon\})$ we define

$$\mathcal{Q}[i, T', v] = \{\mathcal{P} \in \mathcal{Q}^{(i)} \mid W(\mathcal{P}, v) = T', \text{ and if } v \neq \epsilon \text{ then } v \text{ is the final vertex of } \mathcal{P}\}$$

as the set of all potential partial solutions of size i whose set of end vertices is exactly $T' \cup \{v\}$. Observe from this definition that in the case $v = \epsilon$, the last path P_r in each path system $\mathcal{P} = \{P_1, \ldots, P_r\} \in \mathcal{Q}[i, T', \epsilon]$ ends at a "good" vertex; that is, at a terminal vertex which is different from all the end vertices of the other paths $P_1, \ldots, P_{(r-1)}$ in $\mathcal{P}$.

[1] Proof of results labelled with ♣ will appear in the full version of the paper.

It is not difficult to see that this definition of $\mathcal{Q}[i, T', v]$ is a correct notion of a partial solution for Co-Connected T-Join:

▶ **Lemma 3.5.** *Let (G, T, k) be a* Yes *instance of* Co-Connected T-Join *which has a minimal solution of size $k' \leq k$, and let $\ell = \frac{|T|}{2}$. Then for each $1 \leq i \leq k'$ there exist $T' \subseteq T$, $v \in (V(G) \cup \{\epsilon\})$, and path systems $\mathcal{P} = \{P_1, P_2, \ldots, P_r\} \in \mathcal{Q}[i, T', v]$ and $\mathcal{P}' = \{P'_r, P'_{r+1}, \ldots, P'_\ell\}$ in G (where $E(P'_r) = \emptyset$ if $v = \epsilon$) such that (i) $E(\mathcal{P}) \cap E(\mathcal{P}') = \emptyset$, (ii) $P_r P'_r$ is a path in G, and (iii) $\mathcal{P} \cup \mathcal{P}' = \{P_1, P_2, \ldots, P_r P'_r, P'_{r+1}, \ldots, P'_\ell\}$ is an edge-disjoint path system whose edge set is a solution to the instance (G, T, k).*

Proof. Let $\widehat{\mathcal{P}} = \{\widehat{P}_1, \ldots, \widehat{P}_\ell\}$ be a path system in graph G which witnesses – as per Lemma 3.3 – the fact that (G, T, k) has a solution of size k'. If $i = \sum_{j=1}^{r} |E(\widehat{P}_j)|$ for some $1 \leq r \leq \ell$ then the path systems $\mathcal{P} = \{\widehat{P}_1, \widehat{P}_2, \ldots, \widehat{P}_r\} \in \mathcal{Q}[i, T', v]$ and $\mathcal{P}' = \{\emptyset, \widehat{P}_{r+1}, \widehat{P}_{r+2}, \ldots, \widehat{P}_\ell\}$ satisfy the claim, where $T' = T \cap V^e(\mathcal{P})$ and $v = \epsilon$.

If i takes another value then let $1 \leq r \leq \ell$ be such that $\sum_{j=1}^{r-1} |E(\widehat{P}_j)| < i < \sum_{j=1}^{r} |E(\widehat{P}_j)|$. "Split" the path $\widehat{P}_r$ as $\widehat{P}_r = \widehat{P}_r^1 \widehat{P}_r^2$ such that $\sum_{j=1}^{r-1} |E(\widehat{P}_j)| + |E(\widehat{P}_r^1)| = i$. Now the path systems $\mathcal{P} = \{\widehat{P}_1, \widehat{P}_2, \ldots, \widehat{P}_{r-1}, \widehat{P}_r^1\} \in \mathcal{Q}[i, T', v]$ and $\mathcal{P}' = \{\widehat{P}_r^2, \widehat{P}_{r+1}, \widehat{P}_{r+2}, \ldots, \widehat{P}_\ell\}$ satisfy the claim, where $T' = T \cap V^e(\mathcal{P})$ and v is the final vertex of the path $\widehat{P}_r^1$. ◀

Given this notion of a partial solution the natural dynamic programming approach is to try to compute, in increasing order of $1 \leq i \leq k$, partial solutions $\mathcal{Q}[i, T', v]$ for all $T' \subseteq T$, $v \in (V(G) \cup \{\epsilon\})$ at step i. But this is not feasible in polynomial time because the sets $\mathcal{Q}[i, T', v]$ can potentially grow to sizes exponential in $|V(G)|$. Our way out is to observe that to reach a final solution to the problem we do not need to store *every* element of a set $\mathcal{Q}[i, T', v]$ at each intermediate step. Instead, we only need to store a *representative family* $\mathcal{R}$ of partial solutions corresponding to $\mathcal{Q}[i, T', v]$, where $\mathcal{R}$ has the following property: If there is a way of extending – in the sense of Lemma 3.5—any partial solution $\mathcal{P} \in \mathcal{Q}[i, T', v]$ to a final solution then there exists a $\hat{\mathcal{P}} \in \mathcal{R}$ which can be extended the *same* way to a final solution.

Observe now that our final solution and all partial solutions are independent sets in the co-graphic matroid M_G of the input graph G. We use the algorithm of Lokshtanov et al. [13]—see Theorem 2.2—to compute these representative families of potential partial solutions at each intermediate step. In step i of the dynamic programming we store, in place of the set $\mathcal{Q}[i, T', v]$, its $(k-i)$-representative set $\widehat{\mathcal{Q}[i, T', v]} \subseteq^{k-i}_{rep} \mathcal{Q}[i, T', v]$ with respect to the co-graphic matroid M_G; for the purpose of this computation we think of each element $\mathcal{P}$ of $\mathcal{Q}[i, T', v]$ as the *edge set* $E(\mathcal{P})$. Lemma 3.8 below shows that this is a safe step. Whenever we talk about representative families in this section, it is always with respect to the co-graphic matroid M_G associated with G; we do not explicitly mention the matroid from now on. We start with the following definitions.

▶ **Definition 3.6.** Let $1 \leq i \leq k$, $T' \subseteq T, \ell = \frac{|T|}{2}$ and $v \in (V(G) \cup \{\epsilon\})$, and let $\mathcal{Q}[i, T', v]$ be the corresponding set of partial solutions. Let $\mathcal{P} = \{P_1, \ldots, P_r\}$ be a path system in the set $\mathcal{Q}[i, T', v]$. Let $\mathcal{P}' = \{P'_r, P'_{r+1}, \ldots, P'_\ell\}$ be a path system in G (where $E(P'_r) = \emptyset$ if $v = \epsilon$) such that (i) $|E(\mathcal{P}')| \leq (k - i)$, (ii) $P_r P'_r$ is a path in G, (iii) $\mathcal{P} \cup \mathcal{P}' = \{P_1, P_2, \ldots, P_r P'_r, P'_{r+1}, \ldots, P'_\ell\}$ is an edge-disjoint path system that connects disjoint pairs of terminals in T, (iv) $V^e(\mathcal{P} \cup \mathcal{P}') = T$ and (v) $G \setminus (E(\mathcal{P}) \cup E(\mathcal{P}'))$ is connected. Then $\mathcal{P}'$ is said to be an **extender** for $\mathcal{P}$.

▶ **Definition 3.7.** Let $1 \leq i \leq k$, $T' \subseteq T$ and $v \in (V(G) \cup \{\epsilon\})$, and let $\mathcal{Q}[i, T', v]$ be the corresponding set of partial solutions. We say that $\mathcal{J}[i, T', v] \subseteq \mathcal{Q}[i, T', v]$ is a **path-system**

equivalent set to $\mathcal{Q}[i, T', v]$ if the following holds: If $\mathcal{P} \in \mathcal{Q}[i, T', v]$ and $\mathcal{P}'$ be an extender for $\mathcal{P}$, then there exists $\mathcal{P}^* \in \mathcal{J}[i, T', v]$ such that $\mathcal{P}'$ is an extender for $\mathcal{P}^*$ as well. We say that $\mathcal{J}[i, T', v] \sqsubseteq^{k-i}_{peq} \mathcal{Q}[i, T', v]$.

The next lemma shows that a representative family is indeed a path-system equivalent set to $\mathcal{Q}[i, T', v]$.

▶ **Lemma 3.8.** *Let (G, T, k) be an instance of* Co-Connected T-Join *such that the smallest co-connected T-join of G has size k and let $\ell = \frac{|T|}{2}$. Let $1 \leq i \leq k$, $T' \subseteq T$ and $v \in (V(G) \cup \{\epsilon\})$, and let $\mathcal{Q}[i, T', v]$ be the corresponding set of partial solutions. If $\widehat{\mathcal{Q}[i, T', v]} \subseteq^{k-i}_{rep} \mathcal{Q}[i, T', v]$, then $\widehat{\mathcal{Q}[i, T', v]} \sqsubseteq^{k-i}_{peq} \mathcal{Q}[i, T', v]$. More generally, if $\mathcal{J}[i, T', v] \subseteq \mathcal{Q}[i, T', v]$ and $\widehat{\mathcal{J}[i, T', v]} \subseteq^{k-i}_{rep} \mathcal{J}[i, T', v]$ then $\widehat{\mathcal{J}[i, T', v]} \sqsubseteq^{k-i}_{rep} \mathcal{J}[i, T', v]$.*

Proof. We first prove the first claim. The second claim of the lemma follows by similar arguments. Let $\widehat{\mathcal{Q}[i, T', v]} \subseteq^{k-i}_{rep} \mathcal{Q}[i, T', v]$, let $\mathcal{P} = \{P_1, \ldots, P_r\}$ be a path system in the set $\mathcal{Q}[i, T', v]$, and let $\mathcal{P}' = \{P'_r, P'_{r+1}, \ldots, P'_\ell\}$ be a path system in G (where $E(P'_r) = \emptyset$ if $v = \epsilon$) which is an extender for $\mathcal{P}$. We have to show that there exists a path system $\mathcal{P}^* \in \widehat{\mathcal{Q}[i, T', v]}$ such that $\mathcal{P}'$ is an extender for $\mathcal{P}^*$ as well. Since $\mathcal{P}'$ is an extender for $\mathcal{P}$ we have, by definition, that (i) $|E(\mathcal{P}')| \leq (k - i)$, (ii) $P_r P'_r$ is a path in G, (iii) $\mathcal{P} \cup \mathcal{P}' = \{P_1, \ldots, P_r P'_r, P'_{r+1}, \ldots, P'_\ell\}$ is an edge-disjoint path system that connects disjoint pairs of terminals in T, (iv) $V^e(\mathcal{P} \cup \mathcal{P}') = T$ and (v) $G \setminus (E(\mathcal{P}) \cup E(\mathcal{P}'))$ is connected.

Since (i) $\mathcal{P} \in \mathcal{Q}[i, T', v]$, (ii) $E(\mathcal{P}) \cap E(\mathcal{P}') = \emptyset$, (iii) $G \setminus (E(\mathcal{P}) \cup E(\mathcal{P}'))$ is connected, and (iv) $\widehat{\mathcal{Q}[i, T', v]} \subseteq^{k-i}_{rep} \mathcal{Q}[i, T', v]$, there exists a path system $\mathcal{P}^* = \{P^*_1, P^*_2, \ldots, P^*_r\}$ in $\widehat{\mathcal{Q}[i, T', v]}$ such that (i) $E(\mathcal{P}^*) \cap E(\mathcal{P}') = \emptyset$ and (ii) $G \setminus (E(\mathcal{P}^*) \cup E(\mathcal{P}'))$ is connected. This follows directly from the definitions of a co-graphic matroid and a representative set.

We now show that $\mathcal{P}'$ is indeed an extender for $\mathcal{P}^*$. Since $\mathcal{P}$ and $\mathcal{P}^*$ both belong to the set $\mathcal{Q}[i, T', v]$ we get that $|E(\mathcal{P})| = |E(\mathcal{P}^*)| = i$ and that $\mathcal{P}^*$ is an edge-disjoint path system. And since $E(\mathcal{P}^*) \cap E(\mathcal{P}') = \emptyset$, we have that $\mathcal{P}^* \cup \mathcal{P}' = \{P^*_1, \ldots, P^*_{r-1}, P^*_r P'_r, P'_{r+1}, \ldots, P'_\ell\}$ is an edge-disjoint path system but for $P^*_r P'_r$ which could be an *Eulerian walk* (walk where vertices could repeat but not the edges). Now we prove that the "path system" $\mathcal{P}^* \cup \mathcal{P}'$ connects disjoint pairs of terminals in T, but for a pair which is connected by an Eulerian walk. We now consider two cases for the "vertex" v.

Case 1: $v = \epsilon$**.** In this case, since $\mathcal{P}$ and $\mathcal{P}^*$ both belong to the set $\mathcal{Q}[i, T', \epsilon]$ we have that $V^e(\mathcal{P}) = V^e(\mathcal{P}^*) = T'$. Also $E(P'_r) = \emptyset$, and $\mathcal{P} \cup \mathcal{P}'$ is the path system $\{P_1, \ldots, P_r, P'_{r+1}, P'_{r+2}, \ldots, P'_\ell\}$ with exactly $\ell = \frac{|T|}{2}$ paths which connect disjoint pairs of terminals in T. Since $V^e(\mathcal{P} \cup \mathcal{P}') = T$, $\mathcal{P} = \{P_1, \ldots, P_r\}$ and $V^e(\mathcal{P}) = T'$, we get that $V^e(\mathcal{P}') = T \setminus T'$. Now since $V^e(\mathcal{P}^*) = T'$ it follows that $\mathcal{P}^* \cup \mathcal{P}'$ is a path system which connects disjoint pairs of terminals in T.

Case 2: $v \neq \epsilon$**.** In this case, since $\mathcal{P}$ and $\mathcal{P}^*$ both belong to the set $\mathcal{Q}[i, T', v]$ we have that $V^e(\mathcal{P}) = V^e(\mathcal{P}^*) = T' \cup \{v\}$, and that the final vertex of each of these two path systems is v. Also $\mathcal{P} \cup \mathcal{P}' = \{P_1, \ldots, P_r P'_r, P'_{r+1}, P'_{r+2}, \ldots, P'_\ell\}$ is a path system with exactly $\ell = \frac{|T|}{2}$ paths which connect disjoint pairs of terminals in T. Since (i) $V^e(\mathcal{P} \cup \mathcal{P}') = T$, (ii) $\mathcal{P} = \{P_1, \ldots, P_r\}$, (iii) $\mathcal{P}' = \{P'_r, P'_{r+1}, \ldots, P'_\ell\}$, (iv) $V^e(\mathcal{P}) = T' \cup \{v\}$, and (v) the final vertex of the path P_r in $\mathcal{P}$ is v, we get that (i) the initial vertex of the path P'_r in $\mathcal{P}'$ is v and (ii) $V^e(\mathcal{P}') = (T \setminus T') \cup \{v\}$. Now since $V^e(\mathcal{P}^*) = T' \cup \{v\}$ and (ii) the final vertex of $\mathcal{P}^*$ is v it follows that $\mathcal{P}^* \cup \mathcal{P}'$ is a path system which connects disjoint pairs of terminals in T, where $P^*_r P'_r$ which could be an Eulerian walk.

Thus, we have shown that $\mathcal{P}^* \cup \mathcal{P}'$ connects disjoint pairs of terminals in T with paths, except for $P_r^* P_r'$ which could be an Eulerian walk. Combining this with Proposition 3.2 and the fact that $G \setminus (E(\mathcal{P}^*) \cup E(\mathcal{P}'))$ is connected, we get that $E(\mathcal{P}^*) \cup E(\mathcal{P}')$ is a co-connected T-join of G.

Finally, we show that $\mathcal{P}^* \cup \mathcal{P}'$ is a path system. Towards this we only need to show that $P_r^* P_r'$ is not an Eulerian walk but a path. Observe that $|E(\mathcal{P}^*) \cup E(\mathcal{P}')| \leq |E(\mathcal{P}^*)| + |E(\mathcal{P}')| \leq k$. However, $E(\mathcal{P}^*) \cup E(\mathcal{P}')$ is a co-connected T-join of G and thus by our assumption, $E(\mathcal{P}^*) \cup E(\mathcal{P}')$ has size exactly k – thus a minimum sized solution. By Lemma 3.3 this implies that $E(\mathcal{P}^*) \cup E(\mathcal{P}')$ is a forest and hence $P_r^* P_r$ is a path in G. This completes the proof. ◀

For our proofs we also need the transitivity property of the relation $\sqsubseteq_{peq}^{q}$.

▶ **Lemma 3.9 (♣).** *The relation $\sqsubseteq_{peq}^{q}$ is transitive.*

Our algorithm is based on dynamic programming and stores a table $\mathcal{D}[i, T', v]$ for all $i \in \{0, \ldots, k\}$, $T' \subseteq T$ and $v \in V(G) \cup \{\epsilon\}$. The idea is that $\mathcal{D}[i, T', v]$ will *store a path-system equivalent set to* $\mathcal{Q}[i, T', v]$. That is, $\mathcal{D}[i, T', v] \sqsubseteq_{peq}^{k-i} \mathcal{Q}[i, T', v]$. The recurrences for dynamic programming is given by the following.

For $i = 0$, we have the following cases.

$$\mathcal{D}[0, T', v] := \begin{cases} \{\emptyset\} & \text{if } T' = \emptyset \text{ and } v = \epsilon \\ \emptyset & \text{otherwise} \end{cases} \tag{1}$$

For $i \geq 1$, we have the following cases based on whether $v = \epsilon$ or not.

$$\begin{aligned} \mathcal{D}[i, T', v] := \quad & \Bigg(\bigcup_{\substack{t \in T' \\ (t,v) \in E(G)}} \mathcal{D}[i-1, T' \setminus \{t\}, \epsilon] \bullet \{(t, v)\} \Bigg) \bigcup \\ & \Bigg(\bigcup_{(u,v) \in E(G)} \mathcal{D}[i-1, T', u] \bullet \{(u, v)\} \Bigg) \end{aligned} \tag{2}$$

$$\begin{aligned} \mathcal{D}[i, T', \epsilon] := \quad & \Bigg(\bigcup_{\substack{t_1, t_2 \in T' \\ (t_1,t_2) \in E(G)}} \mathcal{D}[i-1, T' \setminus \{t_1, t_2\}, \epsilon] \bullet \{(t_1, t_2)\} \Bigg) \bigcup \\ & \Bigg(\bigcup_{\substack{t \in T' \\ (u,t) \in E(G)}} \mathcal{D}[i-1, T' \setminus \{t\}, u] \bullet \{(u, t)\} \Bigg) \end{aligned} \tag{3}$$

The next lemma will be used in proving the correctness of the algorithm.

▶ **Lemma 3.10.** *For all $i \in \{0, \ldots, k\}, T' \subseteq T, v \in V(G) \cup \{\epsilon\}$, $\mathcal{D}[i, T', v] \sqsubseteq_{peq}^{k-i} \mathcal{Q}[i, T', v]$.*

Proof. Let $\mathcal{I}$ denote the family of independent sets in M_G, the co-graphic matroid associated with G. We prove the lemma using induction on i. The base case is $i = 0$. From the definition of $\mathcal{Q}[0, T', v]$, we have that $\mathcal{Q}[0, T', v] = \{\emptyset\}$ if $T' = \emptyset$ and $v = \epsilon$, and $\mathcal{Q}[0, T', v] = \emptyset$ otherwise.

Now we prove that the claim holds for $i \geq 1$. Let us also assume that by induction hypothesis the claim is true for all $i' < i$. Fix a $T' \subseteq T$, and $v \in V(G) \cup \{\epsilon\}$ and let $\mathcal{Q}[i, T', v]$ be the corresponding set of partial solutions. Let $\mathcal{P} = \{P_1, \ldots, P_r\} \in \mathcal{Q}[i, T', v]$ and $\mathcal{P}' = \{P_r', P_{r+1}', \ldots, P_\ell'\}$ be a path system such that $\mathcal{P}'$ is an extender for $\mathcal{P}$. We need to show that there exists a $\mathcal{P}^* \in \mathcal{D}[i, T', v]$ such that $\mathcal{P}'$ is also an extender for $\mathcal{P}^*$.

Case 1: $v \neq \epsilon$. Consider the path system $\mathcal{P} = \{P_1, \ldots, P_r\} \in \mathcal{Q}[i, T', v]$. $\mathcal{P}$ has i edges and its set of end-vertices is $T' \cup \{v\}$. Also, its final vertex is v. Let (u, v) be the last edge in path P_r. Let P_r'' be the path obtained by deleting edge (u, v) from P_r. More precisely: If P_r has at least two edges then P_r'' is the non-empty path obtained by deleting the edge (u, v) and the vertex v *from* P_r, and if (u, v) is the only edge in P_r (in which case $u \in T'$) then $P_r'' = \emptyset$. Note that the initial vertex of $P_r' \in \mathcal{P}'$ is v. Let uP_r' be the path obtained by concatenating the path uv and P_r'. Let $\mathcal{P}_1 = \{P_1, \ldots, P_r''\}$ and $\mathcal{P}_1' = \{uP_r', P_{r+1}', \ldots, P_\ell'\}$. Then $\mathcal{P}_1$ has $(i-1)$ edges and $\mathcal{P}_1'$ is an extender for $\mathcal{P}_1$. Now we consider two cases:

(u, v) is the only edge in P_r: Here $P_r'' = \emptyset$ and $u \in T'$; let $t = u$. Note that $\mathcal{P}_1 = \{P_1, \ldots, P_{r-1}\} \in \mathcal{Q}[i-1, T' \setminus \{t\}, \epsilon]$. Hence by induction hypothesis there exists $\mathcal{P}_1^* \in \mathcal{D}[i-1, T' \setminus \{t\}, \epsilon]$ such that $\mathcal{P}_1'$ is also an extender for $\mathcal{P}_1^*$. Since $\mathcal{P}_1'$ is an extender for $\mathcal{P}_1^*$, $E(\mathcal{P}_1^*) \cup E(\mathcal{P}_1') \in \mathcal{I}$ (by the definition of extender). This implies that $E(\mathcal{P}_1^*) \cup \{(t, v)\} \in \mathcal{I}$. Since $\mathcal{P}_1^* \in \mathcal{D}[i-1, T' \setminus \{t\}, \epsilon]$ and $(t, v) \in E(G)$, by Equation 2, we get a path system $\mathcal{P}^* \in \mathcal{D}[i, T', v]$ by adding the new path $P_r = tv$ to $\mathcal{P}_1^*$. Since $\mathcal{P}_1'$ is an extender of $\mathcal{P}_1^*$, $\mathcal{P}'$ is an extender of $\mathcal{P}^*$ as well.

(u, v) is not the only edge in P_r: Here $P_r'' \neq \emptyset$, and u is the final vertex in P_r'. Hence $\mathcal{P}_1 = \{P_1, \ldots, P_r''\} \in \mathcal{Q}[i-1, T', u]$. Since $\mathcal{P}_1'$ is an extender for $\mathcal{P}_1$, by induction hypothesis there exists $\mathcal{P}_1^* \in \mathcal{D}[i-1, T', u]$ such that $\mathcal{P}_1'$ is also an extender for $\mathcal{P}_1^*$. By the definition of extender, we have that $E(\mathcal{P}_1^*) \cup E(\mathcal{P}_1') \in \mathcal{I}$. This implies that $E(\mathcal{P}_1^*) \cup \{(u, v)\} \in \mathcal{I}$. Since $\mathcal{P}_1^* \in \mathcal{D}[i-1, T', u]$ and $(u, v) \in E(G)$, by Equation 2, we get a path system $\mathcal{P}^* \in \mathcal{D}[i, T', v]$ by adding the new edge $\{(u, v)\}$ to $\mathcal{P}_1^*$. Since $\mathcal{P}_1'$ is an extender of $\mathcal{P}_1^*$, $\mathcal{P}'$ is an extender of $\mathcal{P}^*$ as well.

Case 2: $v = \epsilon$. We have that $\mathcal{P} = \{P_1, \ldots, P_r\} \in \mathcal{D}[i, T', \epsilon]$. Then $\mathcal{P}$ has i edges, its set of end-vertices is T', and no end-vertex repeats. Let (u, t) be the last edge in path P_r. Then $t \in T'$. Let P_r'' be the path obtained by deleting edge (u, t) from P_r. More precisely: If P_r has at least two edges then P_r'' is the non-empty path obtained by deleting the edge (u, t) and the vertex t *from* P_r, and if (u, t) is the only edge in P_r then $P_r'' = \emptyset$. Let $\mathcal{P}_1 = \{P_1, \ldots, P_r''\}$ and $\mathcal{P}_1' = \{ut, P_r', P_{r+1}', \ldots, P_\ell'\}$. Then $\mathcal{P}_1$ has $(i-1)$ edges and $\mathcal{P}_1'$ is an extender for $\mathcal{P}_1$. Now we consider two cases:

(u, t) is the only edge in P_r: Here $P_r'' = \emptyset$, and $\{u, t\} \subseteq T'$. Let $t_1 = u, t_2 = t$. Then $\mathcal{P}_1$ is a path system in $\mathcal{Q}[i-1, T' \setminus \{t_1, t_2\}, \epsilon]$. By induction hypothesis there exists $\mathcal{P}_1^* \in \mathcal{D}[i-1, T' \setminus \{t_1, t_2\}, \epsilon]$ such that $\mathcal{P}_1'$ is also an extender of $\mathcal{P}_1^*$. By the definition of extender, we have that $E(\mathcal{P}_1^*) \cup E(\mathcal{P}_1') \in \mathcal{I}$. This implies that $E(\mathcal{P}_1^*) \cup \{(t_1, t_2)\} \in \mathcal{I}$. Since $\mathcal{P}_1^* \in \mathcal{D}[i-1, T' \setminus \{t_1, t_2\}, \epsilon]$ and $(t_1, t_2) \in E(G)$, by Equation 3, we get a path system $\mathcal{P}^* \in \mathcal{D}[i, T', v]$ by adding the new path $t_1 t_2$ to $\mathcal{P}_1^*$. Since $\mathcal{P}_1'$ is an extender of $\mathcal{P}_1^*$, $\mathcal{P}'$ is an extender of $\mathcal{P}^*$ as well.

(u, t) is not the only edge in P_r: Here $P_r'' \neq \emptyset$, u is the final vertex in P_r''. Then $\mathcal{P}_1 \in \mathcal{Q}[i-1, (T' \setminus t), u]$. By induction hypothesis there exists $\mathcal{P}_1^* \in \mathcal{D}[i-1, (T' \setminus t), u]$ such that $\mathcal{P}_1'$ is also an extender of $\mathcal{P}_1^*$. By the definition of extender, we have that $E(\mathcal{P}_1^*) \cup E(\mathcal{P}_1') \in \mathcal{I}$. This implies that $E(\mathcal{P}_1^*) \cup \{(u, t)\} \in \mathcal{I}$. Since $\mathcal{P}_1^* \in \mathcal{D}[i-1, (T' \setminus t), u]$ and $(u, t) \in E(G)$, by Equation 3, we get a path system $\mathcal{P}^* \in \mathcal{D}[i, T', \epsilon]$ by adding the new edge (u, t) to $\mathcal{P}_1^*$. Since $\mathcal{P}_1'$ is an extender of $\mathcal{P}_1^*$, $\mathcal{P}'$ is an extender of $\mathcal{P}^*$ as well.

In both cases above we showed that $\mathcal{D}[i, T', v] \sqsubseteq_{peq}^{k-i} \mathcal{Q}[i, T', v]$. ◀

Algorithm, Correctness and Running Time. We now describe the main steps of the algorithm. It finds a smallest sized co-connected T-join (of size at most k) for G. The algorithm iteratively tries to find a solution of size $\frac{|T|}{2} \leq k' \leq k$ and returns a solution corresponding to the smallest k' for which it succeeds; else it returns No. By Lemma 3.8 it is enough, in the dynamic programming (DP) table, to store the representative set $\mathcal{Q}[\widehat{i, T', v}] \subseteq^{k-i}_{rep} \mathcal{Q}[i, T', v]$ instead of the complete set $\mathcal{Q}[i, T', v]$, for all $i \in \{1, 2, \ldots, k\}$, $T' \subseteq T$ and $v \in (V(G) \cup \{\epsilon\})$. In the algorithm we compute and store the set $\mathcal{Q}[\widehat{i, T', v}]$ in the DP table entry $\mathcal{D}[i, T', v]$. We follow Equations 1, 2 and 3 and fill the table $\mathcal{D}[i, T', v]$. For $i = 0$ we use Equation 1 and fill the table. After this we compute the values of $\mathcal{D}[i, T', v]$ in increasing order of i from 1 to k. At the i^{th} iteration of the **for** loop, we compute $\mathcal{D}[i, T', v]$ from the DP table entries computed at the previous iteration. Since we need to keep the size of potential partial solutions in check, we compute the representative family $\mathcal{D}[\widehat{i, T', v}]$ for each DP table entry $\mathcal{D}[i, T', v]$ constructed in the i^{th} iteration and then set $\mathcal{D}[i, T', v] \leftarrow \mathcal{D}[\widehat{i, T', v}]$. By the definition of $\mathcal{Q}[i, T, \epsilon]$ and Lemma 3.4, any path system in $\mathcal{D}[i, T, \epsilon]$ is a solution to the instance (G, T, k); we check for such a solution as the last step. This completes the description of the algorithm.

The correctness of the algorithm follows from the following. By Lemma 3.10 we know that $\mathcal{D}[i, T', v] \sqsubseteq^{k-i}_{peq} \mathcal{Q}[i, T', v]$ and by Lemma 3.8 we have that $\mathcal{D}[\widehat{i, T', v}] \sqsubseteq^{k-i}_{peq} \mathcal{D}[i, T', v]$. Thus, by transitivity of $\sqsubseteq^{q}_{peq}$ (by Lemma 3.9) we have that $\mathcal{D}[\widehat{i, T', v}] \sqsubseteq^{k-i}_{peq} \mathcal{Q}[i, T', v]$. This completes the proof of correctness. We now compute an upper bound on the running time of the algorithm.

▶ **Lemma 3.11.** *The above algorithm runs in time* $\mathcal{O}(2^{(2+\omega)k} \cdot n^2 m^3 k^5) + m^{\mathcal{O}(1)}$ *where* $n = |V(G)|$ *and* $m = |E(G)|$.

Proof. Let $1 \leq i \leq k$ and $T' \subseteq T$ and $v \in (V(G) \cup \{\epsilon\})$ be fixed, and let us consider the running time of computing $\mathcal{D}[\widehat{i, T', v}]$. That is, the running time to compute $(k-i)$-representative family of $\mathcal{D}[i, T', v]$. We know that the co-graphic matroid M_G is representable over $\mathbb{F}_2$ and that its rank is bounded by $m - n + 1$. By Theorem 2.2, the running time of this computation of the $(k-i)$-representative family is bounded by $\mathcal{O}\left(\binom{k}{i} \cdot |\mathcal{D}[i, T', v]| i^3 m^2 + |\mathcal{D}[i, T', v]| \cdot \binom{k}{i}^{\omega-1} (i \cdot m)^{\omega-1}\right) + m^{\mathcal{O}(1)}$.

The family $\mathcal{D}[i, T', v]$ is computed using Equation 2 or Equation 3 from the DP table entries $\mathcal{D}[i-1, T'', u]$, computed in the previous iteration and the size of $\mathcal{D}[i-1, T'', u]$ is bounded according to Theorem 2.2. Thus the size of the family $\mathcal{D}[i, T', v]$ is upper bounded by, $|\mathcal{D}[i, T', v]| \leq ((2k)^2 + 2kn) \cdot \left(\max_{T'' \subseteq T', u \in V} \mathcal{D}[\widehat{i-1, T'', u}]\right)$. Theorem 2.2 gives bounds on the sizes of these representative families $\mathcal{D}[\widehat{i-1, T'', u}]$, from which we get $|\mathcal{D}[i, T', v]| \leq 4kn \cdot mi\binom{k}{i-1}$. Observe that since the number choices for (T', v) such that $T' \subseteq T$ and $v \in V(G)\{\epsilon\}$ is bounded by $4^k(n+1)$, and we compute DP table entries for $i = 1$ to k, the overall running time can be bounded by $\mathcal{O}\Big(4^k n \sum_{i=1}^{k} \Big(\binom{k}{i} \cdot \binom{k}{i-1} kni^4 m^3 + \binom{k}{i-1} \cdot \binom{k}{i}^{\omega-1} kn(im)^{\omega}\Big)\Big) + m^{\mathcal{O}(1)}$. The running time above simplifies to $\mathcal{O}(2^{(2+\omega)k} \cdot n^2 m^3 k^5) + m^{\mathcal{O}(1)}$. ◀

Putting all these together we get

▶ **Theorem 3.12.** Co-Connected T-Join *can be solved in* $\mathcal{O}(2^{(2+\omega)k} \cdot n^2 m^3 k^6) + m^{\mathcal{O}(1)}$ *time where* $n = |V(G)|$ *and* $m = |E(G)|$.

Using Theorem 3.1 and Theorem 3.12 we get

▶ **Theorem 3.13.** Undirected Eulerian Edge Deletion *can be solved in time* $\mathcal{O}(2^{(2+\omega)k} \cdot n^2m^3k^6) + m^{\mathcal{O}(1)}$ *where* $n = |V(G)|$ *and* $m = |E(G)|$.

4 Directed Eulerian Edge Deletion

In this section we modify the algorithm described for Undirected Eulerian Edge Deletion to solve the directed version of the problem. The main ingredient of the proof is the characterization of "solution" for the directed version of the problem. We begin with a few definitions. For a digraph D, we call $S \subseteq A(D)$ a *balanced arc deletion set*, if $D \setminus S$ is balanced. We call a set $S \subseteq A(D)$ a *co-connected balanced arc deletion set* if $D \setminus S$ is balanced and weakly connected.

Let (D, k) be an instance to Directed Eulerian Edge Deletion. A solution $S \subseteq A(D)$ of the problem should satisfy the following two properties, (a) S must be a balanced arc deletion set of D and, (b) $D \setminus S$ must be strongly connected. In fact, something stronger is known in the literature.

▶ **Proposition 4.1** ([1]). *A digraph D is Eulerian if and only if D is weakly connected and balanced.*

Due to Proposition 4.1, we can relax the property (b) of the solution S and replace the requirement of having $D \setminus S$ as strongly connected with just requiring $D \setminus S$ to be be weakly connected. Now observe that solution S of Directed Eulerian Edge Deletion is in fact a co-connected balanced arc deletion set of the directed graph D. Thus our goal is to compute a minimal co-connected balanced arc deletion set of D of size at most k.

We start with the following easy property of in-degrees and out-degrees of vertices in D. For a digraph D, define $\mathcal{T}^- = \{v \in V(D) \mid d_D^-(v) > d_D^+(v)\}$, $\mathcal{T}^= = \{v \in V(D) \mid d_D^-(v) = d_D^+(v)\}$ and $\mathcal{T}^+ = \{v \in V(D) \mid d_D^-(v) < d_D^+(v)\}$.

▶ **Proposition 4.2 (♣).** *In a digraph D,* $\sum_{v \in \mathcal{T}^+} d_D^+(v) - d_D^-(v) = \sum_{v \in \mathcal{T}^-} d_D^-(v) - d_D^+(v)$.

The following lemma characterizes the set of arcs which form a minimal solution S of the given instance (D, k). We then use this characterization to design a dynamic-programming algorithm for the problem.

▶ **Lemma 4.3 (♣).** *Let D be a digraph, and $\ell = \sum_{v \in \mathcal{T}^+} d_D^+(v) - d_D^-(v)$. Let $S \subseteq A(D)$ be a minimal co-connected balanced arc deletion set. Then S is a union of ℓ arc disjoint paths $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ such that*

1. *For $i \in \{1, \ldots, \ell\}$, P_i starts at a vertex in $\mathcal{T}^+$ and ends at a vertex in $\mathcal{T}^-$.*
2. *The number of paths in $\mathcal{P}$ that starts at $v \in \mathcal{T}^+$ is equal to $d_D^+(v) - d_D^-(v)$ and the number of paths in $\mathcal{P}$ that ends at $u \in \mathcal{T}^-$ is equal to $d_D^-(u) - d_D^+(u)$.*

Finally, we prove a kind of "converse" of Lemma 4.3.

▶ **Lemma 4.4 (♣).** *Let D be a digraph, $\ell = \sum_{v \in \mathcal{T}^+} d_D^+(v) - d_D^-(v)$ and let $S \subseteq A(D)$. Furthermore, S is a union of ℓ arc disjoint paths $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ with the following properties.*

1. *The digraph $D \setminus S$ is weakly connected.*
2. *For $i \in \{1, \ldots, \ell\}$, P_i starts at a vertex in $\mathcal{T}^+$ and ends at a vertex in $\mathcal{T}^-$.*
3. *The number of paths in $\mathcal{P}$ that starts at $v \in \mathcal{T}^+$ is equal to $d_D^+(v) - d_D^-(v)$ and the number of paths in $\mathcal{P}$ that ends at $u \in \mathcal{T}^-$ is equal to $d_D^-(u) - d_D^+(u)$.*

Then S is a co-connected balanced arc deletion set.

Now we are ready to describe the algorithm for Directed Eulerian Edge Deletion. Let (D, k) be an instance of the problem. Lemma 4.3 and Lemma 4.4 imply that for a solution we can seek a path system $\mathcal{P}$ with properties mentioned in Lemma 4.4. Let $\mathcal{T}_m^+$ be the multiset of vertices in the graph G such that each vertex $v \in \mathcal{T}^+$ appears $d_D^+(v) - d_D^-(v)$ times in $\mathcal{T}_m^+$. Similarly, let $\mathcal{T}_m^-$ be the multiset of vertices in the graph D such that each vertex $v \in \mathcal{T}^-$ appears $d_D^-(v) - d_D^+(v)$ times in $\mathcal{T}_m^-$. Due to Proposition 4.2 we know that $|\mathcal{T}_m^+| = |\mathcal{T}_m^-|$. Observe that if $|\mathcal{T}_m^+| > k$, then any balanced arc deletion set must contain more than k arcs and thus the given instance is a No instance. So we assume that $|\mathcal{T}_m^+| \leq k$.

Lemma 4.3 implies that the solution can be thought of as a path system $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ connecting vertices from $\mathcal{T}_m^+$ to the vertices of $\mathcal{T}_m^-$ such that all the vertices of $\mathcal{T}_m^+ \cup \mathcal{T}_m^-$ appear as end points exactly once and $D \setminus A(\mathcal{P})$ is weakly connected. Observe that the solution is a path system with properties which are similar to those in the undirected case of the problem. Indeed, the solution S corresponds to an independent set in the co-graphic matroid of the underlying (undirected) graph of D. After this the algorithm for Directed Eulerian Edge Deletion is identical to the algorithm for Co-Connected T-Join. Let $T = \mathcal{T}_m^+ \cup \mathcal{T}_m^-$. We can define a notion of partial solutions analogous to $\mathcal{Q}[i, T', v]$. The definition of *extender* remains the same except for the last item, where we now require that $\mathcal{P} \cup \mathcal{P}'$ is an arc disjoint path system connecting vertices from $\mathcal{T}_m^+$ to the vertices of $\mathcal{T}_m^-$ such that every vertex of $\mathcal{T}_m^+ \cup \mathcal{T}_m^-$ is an endpoint of exactly one path. Finally, we can define the recurrences for dynamic programming similar to those defined for $\mathcal{D}[i, T', v]$ in the case of Co-Connected T-Join. We then use these recurrences along with an algorithm to compute representative families to solve the given instance. The correctness of the algorithm follows via similar arguments as before. And by an analysis similar to the case of Co-Connected T-Join we can obtain the following bound on the running time of the algorithm.

▶ **Theorem 4.5.** Directed Eulerian Edge Deletion *can be solved in time* $\mathcal{O}(2^{(2+\omega)k} \cdot n^2 m^3 k^6) + m^{\mathcal{O}(1)}$ *where where* $n = |V(D)|$ *and* $m = |A(D)|$.

References

1 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

2 Leizhen Cai and Boting Yang. Parameterized complexity of even/odd subgraph problems. *J. Discrete Algorithms*, 9(3):231–240, 2011.

3 Marek Cygan, Fedor Fomin, Bart M. P. Jansen, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Open Problems for FPT School 2014, Będlewo, Poland. `http://fptschool.mimuw.edu.pl/opl.pdf`.

4 Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Ildikó Schlotter. Parameterized complexity of eulerian deletion problems. *Algorithmica*, 68(1):41–61, 2014.

5 Konrad K Dabrowski, Petr A Golovach, Pim van't Hof, and Daniël Paulusma. Editing to eulerian graphs. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science(FSTTCS)*, 2014.

6 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2010.

7 Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

8 Fedor V. Fomin and Petr A. Golovach. Long circuits and large euler subgraphs. In *ESA*, volume 8125, pages 493–504, 2013.

9 Fedor V. Fomin and Petr A. Golovach. Parameterized complexity of connected even/odd subgraph problems. *J. Comput. Syst. Sci.*, 80(1):157–179, 2014.

10 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *SODA*, pages 142–151, 2014.

11 András Frank. A survey on T-joins, T-cuts, and conservative weightings. In *Combinatorics, Paul Erdös is eighty*, volume 2, pages 213–252. János Bolyai Mathematical Society, 1993.

12 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*, pages 450–459. IEEE, 2012.

13 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. In *Proceedings of the Automata, Languages, and Programming – 42nd International Colloquium, (ICALP 2015)*, pages 922–934, 2015.

14 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci*, 410(44):4471–4479, 2009.

15 B. Monien. How to find long paths efficiently. *Ann. Discrete Math.*, 25:239–254, 1985.

16 James G Oxley. *Matroid theory*, volume 3. Oxford University Press, 2006.

17 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC 2012)*, pages 887–898. ACM, 2012.

The Parameterized Complexity of the Minimum Shared Edges Problem*

Till Fluschnik[1], Stefan Kratsch[2], Rolf Niedermeier[1], and Manuel Sorge[1]

1 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany,** {till.fluschnik, rolf.niedermeier, manuel.sorge}@tu-berlin.de
2 **Institut für Informatik, Universität Bonn, Germany,** kratsch@cs.uni-bonn.de

Abstract

We study the NP-complete Minimum Shared Edges (MSE) problem. Given an undirected graph, a source and a sink vertex, and two integers p and k, the question is whether there are p paths in the graph connecting the source with the sink and sharing at most k edges. Herein, an edge is shared if it appears in at least two paths. We show that MSE is W[1]-hard when parameterized by the treewidth of the input graph and the number k of shared edges combined. We show that MSE is fixed-parameter tractable with respect to p, but does not admit a polynomial-size kernel (unless NP $\subseteq$ coNP/poly). In the proof of the fixed-parameter tractability of MSE parameterized by p, we employ the treewidth reduction technique due to Marx, O'Sullivan, and Razgon [ACM TALG 2013].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Parameterized complexity, kernelization, treewidth, treewidth reduction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.448

1 Introduction

We consider the parameterized complexity of the following basic routing problem.

Minimum Shared Edges (MSE)
Input: An undirected graph $G = (V, E)$, $s, t \in V$, $p \in \mathbb{N}$ and $k \in \mathbb{N}_0$.
Question: Is there a (p, s, t)-routing in G in which at most k edges are shared?

Herein, a (p, s, t)-*routing* is a cardinality-p set of s-t paths, and an edge is called *shared* if it is contained in at least two of the paths in the routing. If s and t are understood from the context, we simplify notation and speak of p-routings and refer to the paths it contains as *routes*. Minimum Shared Edges is polynomial-time solvable with $k = 0$, while it becomes NP-hard for general values of k [14].

Minimum Shared Edges has two natural applications. One is to route an important person which is under threat of attack from s to t in a street network. In order to confound attackers, $p - 1$ additional, empty convoys are routed, and guards are placed on streets that are shared by routes. Minimum Shared Edges then minimizes the costs to place guards [20]. A second application arises from finding a resilient way of communication

* Due to space constraints numerous details are deferred to a full version. Till Fluschnik, Stefan Kratsch, and Manuel Sorge gratefully acknowledge support by the DFG, projects DAMM (NI 369/13-2), PREMOD (KR 4286/2-1), and DAPA (NI 369/12-2), respectively.

between two servers s and t in an interconnection network, assuming that $p-1$ faulty connections may be present that block or alter the communicated information. Finding p edge-disjoint paths ensures at least one piece of information arrives unscathed. When this is not possible, and if we can ensure that a link is not faulty by expending some fixed cost per link, then Minimum Shared Edges is the problem of finding a resilient way of communication that minimizes the overall costs [21].

We study Minimum Shared Edges from a parameterized complexity perspective, that is, for certain parameters ℓ of the inputs (of size r), we identify algorithms with running time $f(\ell) \cdot \text{poly}(r)$ or we prove that such algorithms are unlikely to exist. There are two natural parameters for Minimum Shared Edges: the number p of routes and the number k of shared edges. Both of them can be reasonably assumed to be small in applications. As we will see, there is also a connection between p and the treewidth tw of G.

Related Work. Omran et al. [20] introduced Minimum Shared Edges on directed graphs and showed NP-hardness by a reduction from Set Cover. The reduction also implies W[1]-hardness with respect to the number k of shared arcs in this directed case. Undirected Minimum Shared Edges admits an XP-algorithm with respect to treewidth, more specifically, it can be solved in $O((n+m) \cdot (p+1)^{2^{\omega \cdot (\omega+1)/2}})$ time [24], where ω upper-bounds the treewidth of the input graph.

Assadi et al. [2] introduced a generalization of directed Minimum Shared Edges, called Minimum Vulnerability, which additionally considers arc weights (the cost of sharing an arc), arc capacities (an upper bound on the number of routes supported by an arc) and a share-threshold for each arc (the threshold of routes, possibly other than two, after which the arc becomes shared). Directed Minimum Vulnerability admits an XP-algorithm with respect to the number p of routes [2]. Undirected Minimum Vulnerability is NP-hard even on bipartite series-parallel graphs, but admits a pseudo-polynomial-time algorithm on bounded treewidth graphs [1]. Furthermore, Minimum Vulnerability is fixed-parameter tractable with respect to p on chordal graphs [1].

There are also several results regarding approximation algorithms and lower bounds [2, 20]; however, our focus is on exact algorithms.

Our Contributions. We present two main results: Minimum Shared Edges is fixed-parameter tractable (FPT) with respect to the number p of routes and it is W[1]-hard with respect to the treewidth tw and the number k of shared edges combined. Moreover, complementing the fixed-parameter tractability result with respect to p, we show that there is no polynomial-size problem kernel with respect to p (Section 5), unless NP $\subseteq$ coNP/poly.

The FPT result with respect to p is obtained by modifying the input graph so that the resulting graph has treewidth bounded by some (exponential) function of p using the treewidth reduction technique [17] (see Section 4). Then we apply a dynamic program which also is an FPT algorithm with respect to p and tw (Section 3). For this purpose, we design a new dynamic program rather than using the ones from the literature [1, 2]. In comparison, ours yields an improved running time in the FPT algorithm with respect to p, that is, the dependence is doubly exponential on p rather than triply exponential. Our result complements the known FPT algorithm for undirected Minimum Vulnerability on chordal graphs, parameterized by p [1]. Treewidth reduction has lately also found applications in a wide variety of other problems, for example, in graph coloring [5], graph partitioning [3], and arc routing [15].

As mentioned, our second main result is that Minimum Shared Edges is W[1]-hard with respect to the treewidth tw and the number k of shared edges combined. This provides a

corresponding lower bound for the known polynomial-time algorithms on constant-treewidth graphs for Minimum Shared Edges and for the more general undirected Minimum Vulnerability [1, 2]. More precisely, the degree of the polynomial in the running time depend on tw and our result shows that removing this dependence is impossible unless FPT = W[1]. Interestingly, the known dynamic programs on tree decompositions keep track of the number of routes over certain separators in their tables. Our hardness result shows that information of this sort is crucial.

2 Preliminaries

We use basic notation from parameterized complexity [6, 10, 13, 18] and graph theory [8, 23].

Graphs and Tree Decompositions. Unless stated otherwise, all graphs are without parallel edges or loops. When it is not ambiguous, we use n for the number of vertices of a graph and m for the number of edges.

Let $G = (V, E)$ be an undirected graph. We write $V(G)$ for the vertex set of graph G and $E(G)$ for the edge set of graph G. We define the size of graph G as $|G| := |V(G)|+|E(G)|$. For a vertex set $W \subseteq V(G)$, we denote by $G[W]$ the subgraph of G *induced* by the vertex set W. For an edge set $F \subseteq E(G)$, we denote by $G[F]$ the subgraph of G *induced* by the edge set F. We write G/F and $G\backslash F$ for the contraction and the deletion of the edges in F, respectively (we write G/e and $G\backslash e$ for short if $F = \{e\}$).

A *tree decomposition* of a graph G is a tuple $\mathbb{T} := (T, (B_\alpha)_{\alpha \in V(T)})$ of a tree T and family $(B_\alpha)_{\alpha \in V(T)}$ of sets $B_\alpha \subseteq V(G)$, called *bags*, such that

(i) $V(G) = \bigcup_{\alpha \in V(T)} B_\alpha$,
(ii) for every edge $e \in E(G)$ there exists an $\alpha \in V(T)$ such that $e \subseteq B_\alpha$ and
(iii) for each $v \in V(G)$, the graph induced by the node set $\{\alpha \in V(T) \mid v \in B_\alpha\}$ is a tree.

The *width* ω of a tree decomposition $\mathbb{T}$ of a graph G is defined as $\omega(\mathbb{T}) := \max\{|B_\alpha|-1 \mid \alpha \in V(T)\}$. The *treewidth* $\mathsf{tw}(G)$ of a graph G is the minimum width over all tree decompositions of G. A tree decomposition $\mathbb{T} = (T, (B_\alpha)_{\alpha \in V(T)})$ is a *nice tree decomposition with introduce edge nodes* if the following conditions hold.

(i) The tree T is rooted and binary.
(ii) For each edge in $E(G)$ there is exactly one *introduce edge node* in $\mathbb{T}$, where an introduce edge node is a node α in the tree decomposition $\mathbb{T}$ of G labeled with an edge $\{v, w\} \in E(G)$ with $v, w \in B_\alpha$ that has exactly one child node α'; furthermore $B_\alpha = B_{\alpha'}$.
(iii) Each node $\alpha \in V(T)$ is of one of the following types:
- introduce edge node;
- *leaf node*: α is a leaf of T and $B_\alpha = \emptyset$;
- *introduce vertex node*: α is an inner node of T with exactly one child node $\beta \in V(T)$; furthermore $B_\beta \subseteq B_\alpha$ and $|B_\alpha\backslash B_\beta| = 1$;
- *forget node*: α is an inner node of T with exactly one child node $\beta \in V(T)$; furthermore $B_\alpha \subseteq B_\beta$ and $|B_\beta\backslash B_\alpha| = 1$;
- *join node*: α is an inner node of T with exactly two child nodes $\beta, \gamma \in V(T)$; furthermore $B_\alpha = B_\beta = B_\gamma$.

A given tree decomposition can be modified in linear time to fulfill the above constraints; moreover, the number of nodes in such a tree decomposition of width ω is $O(\omega \cdot n)$ [16, 7].

Cuts and Paths. Let G be an undirected, connected graph. A *cut* $C \subseteq E$ is a set of edges such that the graph $G\backslash C$ is not connected. Let $s, t \in V(G)$ be two vertices in G. An *s-t cut* C

is a cut such that the vertices s and t are not connected in $G \backslash C$. A *minimum s-t cut* is an s-t cut C such that $|C| = \min |C'|$, where the minimum is taken over all s-t cuts C' in G. An s-t cut C in G is *minimal* if for all edges $e \in C$ it holds that $C \backslash \{e\}$ is not an s-t cut in G.

A *path* is a connected graph with exactly two vertices of degree one and no vertex of degree at least three. We call the vertices with degree one the *endpoints* of the path. The *length* of a path is defined as the number of edges in the path. For two distinct vertices $s, t \in V(G)$, we refer to the path with endpoints s and t (as subgraph of G) as s-t path in G.

Parameterized Complexity. A *parameterized problem* is a set of instances $(\mathcal{I}, \ell)$, where $\mathcal{I} \in \Sigma^*$ for a finite alphabet Σ, and $\ell \in \mathbb{N}$ is the *parameter*. A parameterized problem Q is *fixed-parameter tractable*, shortly FPT, if there exists an algorithm that on input $(\mathcal{I}, \ell)$ decides whether $(\mathcal{I}, \ell)$ is a yes-instance of Q in $f(\ell) \cdot |\mathcal{I}|^{O(1)}$ time, where f is a computable function independent of $|\mathcal{I}|$.

W$[t]$, $t \geq 1$, are classes that (amongst others) contain parameterized problems which presumably do not admit FPT algorithms. Hardness for W$[t]$ can be shown by reducing from a W$[t]$-hard problem, using a *parameterized* reduction, that is, a many-to-one reduction that runs in FPT time and maps any instance $(\mathcal{I}, \ell)$ to another instance $(\mathcal{I}', \ell')$ such that $\ell' \leq f(\ell)$ for some computable function f.

A parameterized problem Q is *kernelizable* if there exists a polynomial-time self-reduction that maps an instance $(\mathcal{I}, \ell)$ of Q to another instance $(\mathcal{I}', \ell')$ of Q such that: (1) $|\mathcal{I}'| \leq \lambda(\ell)$ for some computable function λ, (2) $\ell' \leq \lambda(\ell)$, and (3) $(\mathcal{I}, \ell)$ is a yes-instance of Q if and only if $(\mathcal{I}', \ell')$ is a yes-instance of Q. The instance $(\mathcal{I}', \ell')$ is called the *problem kernel* of $(\mathcal{I}, \ell)$ and λ is called its *size*.

3 An Algorithm for Small Treewidth and Small Number of Paths

In this section we present the following theorem.

▶ **Theorem 1.** *Let G be a graph with $s, t \in V(G)$ given together with a tree decomposition of width ω. Let $p \in \mathbb{N}$ be an integer. Then the minimum number of shared edges in a (p, s, t)-routing can be computed in $O(p \cdot (\omega + 4)^{3 \cdot p \cdot (\omega+3)+4} \cdot n)$ time.*

The proof is based on a dynamic program that computes a table for each node of the (arbitrarily rooted) tree decomposition in a bottom-up fashion. For our application, it is convenient to use a nice tree decomposition with introduce edge nodes such that each bag contains the sink and the source node. For each node α in the tree decomposition $\mathbb{T}$ of G, we define V_α as the set of vertices and E_α as the set of edges that are introduced in the subtree rooted at node α. In other words, a vertex $v \in V(G)$ is in V_α if and only if there exists at least one introduce vertex node in the subtree rooted at node α that introduced vertex v. As a special case, since the vertices s and t are contained in every bag, we consider s and t as introduced by each leaf node. An edge $e \in E(G)$ is in E_α if and only if there exists an introduce edge node in the subtree rooted at node α that introduced edge e. Recall that there is a unique introduce edge node for every edge of graph G. We define $G_\alpha := (V_\alpha, E_\alpha)$ as the graph for node α. For every leaf node α in $\mathbb{T}$, we set $V_\alpha = \{s, t\}$ and $E_\alpha = \emptyset$.

Partial Solutions. We define a set of p forests in G_α as a *partial solution* L_α for node α. Instead of asking for p s-t routes that share at most k edges, we can ask for p s-t forests that share at most k edges, where an s-t forest is a forest that contains at least one tree connecting

vertices s and t. Note that every forest that contains a tree containing both vertices s and t can be "reduced" to an s-t path. A partial solution L_α has a cost value $c(L_\alpha)$, which is the number of edges in G_α that appear in at least two of the p forests in L_α.

In order to represent the intersection of the trees in a partial solution with the bag that we are currently considering, we use the following notation. For each node α in the tree decomposition $\mathbb{T}$ of G, we consider p-tuples of pairs $\mathcal{X}^\alpha := (\mathcal{Y}_q^\alpha, Z_q^\alpha)_{q=1,\ldots,p}$, where for each $q \in [p]$, $Z_q^\alpha \subseteq B_\alpha$ together with $\mathcal{Y}_q^\alpha \subseteq 2^{B_\alpha}$ is a partition of B_α, that is,

(i) $\bigcup_{M \in \mathcal{Y}_q^\alpha} M \cup Z_q^\alpha = B_\alpha$, and

(ii) for all $X, Y \in \mathcal{Y}_q^\alpha \cup \{Z_q^\alpha\}$ with $X \neq Y$ it holds that $X \cap Y = \emptyset$.

We say that $\mathcal{X}^\alpha$ is a *signature* for node α. For each $q \in [p]$, we call the pair $(\mathcal{Y}_q^\alpha, Z_q^\alpha)$ a *segmentation* of the vertex set B_α. We write segmentation q instead of segmentation with index q for short. We call each $M \in \mathcal{Y}_q^\alpha$ a *segment* of the segmentation q and we call Z_q^α the *zero*-segment of the segmentation q.

To connect signatures (and segmentations) with the partial solutions that they represent, we use the following notation. We say that the signature $\mathcal{X}^\alpha$ is a *valid* signature for node α if there is a partial solution L_α for node α such that for each $q \in [p]$, the zero-segment Z_q^α is the set of nodes in B_α that do not appear in the forest with index q and for each set $M \in \mathcal{Y}_q^\alpha$, there is a tree S in the forest with index q such that $M = B_\alpha \cap V(S)$. In other words, the sets in $\mathcal{Y}_q^\alpha$ correspond to connected components in the forest with index q of the partial solution. We say that $\mathcal{X}^\alpha$ is a signature *induced* by the partial solution L_α if $\mathcal{X}^\alpha$ is a valid signature for node α and the partial solution L_α validates $\mathcal{X}^\alpha$. In this case, for each $q \in [p]$, the pair $(\mathcal{Y}_q^\alpha, Z_q^\alpha)$ is an *induced* segmentation. We remark that given $\mathcal{X}^\alpha$, there can be exactly one, more than one, or no partial solution with signature $\mathcal{X}^\alpha$. Given a partial solution L_α for G_α, there is exactly one signature induced by L_α. Let $\mathcal{X}^\alpha$ be a signature for node α such that there is no partial solution for G_α that induces the signature $\mathcal{X}^\alpha$, then we say that $\mathcal{X}^\alpha$ is an *invalid* signature.

Let $\mathbb{T} = (T_\mathbb{T}, (B_\alpha)_{\alpha \in V(T_\mathbb{T})})$ be a nice tree decomposition of G with introduce edge nodes and vertices s and t contained in every bag. Let $\omega := \omega(\mathbb{T})$ be the width of $\mathbb{T}$. We consider the table T in the following dynamic program that we process bottom-up on the tree decomposition $\mathbb{T}$, that is, we start to fill the entries of the table T at the leaf nodes of the tree decomposition $\mathbb{T}$ and we traverse the tree of the tree decomposition from the leaves to the root. For a node α in the tree decomposition $\mathbb{T}$ and a signature $\mathcal{X}^\alpha$ for node α, the entry $T[\alpha, \mathcal{X}^\alpha]$ is defined as

$$T[\alpha, \mathcal{X}^\alpha] := \begin{cases} \min c(L_\alpha), & \text{if } \mathcal{X}^\alpha \text{ is a valid signature,} \\ \infty, & \text{otherwise,} \end{cases}$$

where the minimum is taken over all partial solutions L_α in G_α such that L_α induces the signature $\mathcal{X}^\alpha$.

For each type of node in $\mathbb{T}$, we define a rule on how to fill each entry in T, and discuss the running time for applying the rule and the running time for filling all entries in T for the given type of node. Due to space constraints, we give some details only for introduce edge nodes, and defer the correctness proof and the remaining nodes to the full version of the paper.

Introduce Edge Node. Let α be an introduce edge node of $\mathbb{T}$, let β be the child node of α, and let $e = \{v, w\}$ be the edge introduced by node α. Two signatures $\mathcal{X}^\alpha$ and $\mathcal{X}^\beta$ are *compatible* if for each $q \in [p]$, one of the following conditions holds:

(i) $\mathcal{Y}_q^\alpha = \mathcal{Y}_q^\beta$, or

(ii) $\mathcal{Y}_q^\alpha = (\mathcal{Y}_q^\beta \backslash \{M_1, M_2\}) \cup \{M_1 \cup M_2\}$ with $M_1, M_2 \in \mathcal{Y}_q^\beta$, $M_1 \neq M_2$, and $v \in M_1$ and $w \in M_2$.

If $\mathcal{X}^\alpha$ and $\mathcal{X}^\beta$ are compatible, then let $Q \subseteq [p]$ be the set of indices such that for all $q \in Q$ (ii) holds and for all $q \in [p] \backslash Q$ (i) holds. We say that $\mathcal{X}^\alpha$ and $\mathcal{X}^\beta$ are *share-compatible* if $|Q| \geq 2$. We claim that

$$T[\alpha, \mathcal{X}^\alpha] = \min_{\mathcal{X}^\beta \text{ compatible with } \mathcal{X}^\alpha} \left(T[\beta, \mathcal{X}^\beta] + \begin{cases} 1, & \text{if } \mathcal{X}^\beta \text{ and } \mathcal{X}^\alpha \text{ are share-compatible,} \\ 0, & \text{otherwise.} \end{cases} \right)$$

In other words, two signatures $\mathcal{X}^\alpha$ for node α and $\mathcal{X}^\beta$ for node β are compatible if and only if for all $q \in [p]$, either by (i) it holds that the segmentation q in $\mathcal{X}^\alpha$ is equal to the segmentation q of $\mathcal{X}^\beta$, or by (ii) it holds that the segmentation q of $\mathcal{X}^\alpha$ is the result of merging two segments in the segmentation q of $\mathcal{X}^\beta$, where none of the two segments is the zero-segment, and vertex v is in the one segment, and vertex w is in the other segment. This corresponds to connecting two trees by edge e in the forest with index q, where v is in the one tree and w in the other tree. Note that connecting two vertex-disjoint trees by exactly one edge yields a tree. The deletion of edge e in every forest of a partial solution for G_α that includes the edge e yields a partial solution for G_β. We remark that $G_\alpha = G_\beta + \{e\}$, that is, G_α differs from G_β only by the additional edge e.

Running time. For each signature $\mathcal{X}^\alpha$, we check all signatures $\mathcal{X}^\beta$ for node β for compatibility, that means, we need to check for each $q \in [p]$ whether the segmentations are equal (i) or whether the segmentation q of $\mathcal{X}^\alpha$ is derived by merging two segments in the segmentation q of $\mathcal{X}^\beta$ (ii). To check condition (i) as well as to check condition (ii) can be done in $O(p \cdot |B_\alpha|^2)$ time. Therefore, the overall running time for filling all entries in T for an introduce edge node is in $O(p \cdot (\omega + 2)^{2 \cdot p \cdot (\omega+1)+2})$.

The bottleneck in computing the tables is in the join nodes; they induce a running time portion of $O(p \cdot (\omega + 2)^{3 \cdot p \cdot (\omega+1)+3})$. Hence, filling the tables for each node in the tree decomposition can be done in the running time claimed by Theorem 1. By the above arguments about partial solutions, the minimum number of shared edges in a (p, s, t)-routing can then be read off from the table in the root node of the tree decomposition, where we take the minimum value over all signatures where for each of the p segmentations there exists a segment that contains both vertices s and t. Hence, Theorem 1 follows.

We remark that we can modify the dynamic program in such a way that we can solve the weighted variant of Minimum Shared Edges, that is, with weights $w : E(G) \to \mathbb{N}$ on the edge set of the input graph.

4 Fixed-Parameter Tractability with Respect to the Number of Paths

In this section we outline a proof for the following classification result.

▶ **Theorem 2.** Minimum Shared Edges *is fixed-parameter tractable with respect to the number p of routes.*

The basic idea for the proof is to use treewidth reduction [17], a way to process a graph G containing terminals s, t in such a way that each minimal s-t separator of size at most $p-1$ is preserved and the treewidth of the resulting graph is bounded by a function of p. The reason that this approach works is that each (p, s, t)-routing is characterized by its shared edges, and these are contained in minimal cuts of size at most $p-1$. However, treewidth reduction preserves only minimal separators, that is, vertex sets, and not necessarily minimal cuts,

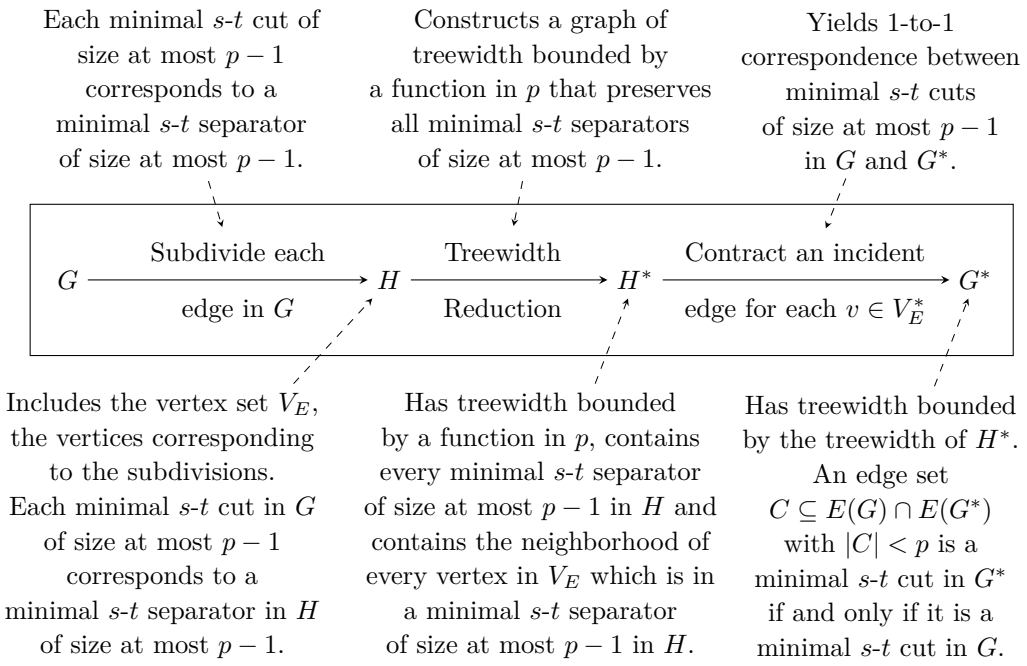


Figure 1 Overview of the strategy behind the proof of Theorem 2.

that is, edge sets. Hence, we need to further process the input graph and the graph coming out of the treewidth reduction process.

We now describe the approach in more detail; refer to Figure 1 for an overview of the following modifications and the graphs obtained in each step. In the following, we modify step by step graph G to graph G^*. We start with the following lemma which states that if our instance is a yes-instance, then we can find a solution where each of the shared edges is part of a minimal s-t cut of size smaller than the number p of routes.

▶ **Lemma 3.** *If (G, s, t, p, k) is a yes-instance of* MSE *and G has a minimal s-t cut of size smaller than p, then there exists a solution $F \subseteq E$ such that each $e \in F$ is in a minimal s-t cut of size smaller than p in G.*

Recall that if G does not have a minimal s-t cut of size smaller than p, then we can find p s-t routes without sharing an edge.

As mentioned before, as part of our approach we use the treewidth reduction technique [17]. Given a graph $G = (V, E)$ with $T = \{s, t\} \subseteq V(G)$ and an integer $\ell \in \mathbb{N}$, first the treewidth reduction technique computes the set C of vertices containing all vertices in G which are part of a minimal s-t separator of size at most ℓ in G. Then, it constructs the so-called *torso* of graph G given C and T, that is, the induced subgraph $G[C \cup T]$ with additional edges between each pair of vertices $v, w \in C \cup T$ with $\{v, w\} \notin E(G)$ if there is a v-w path in G whose internal vertices are not contained in $C \cup T$. Finally, each of these additional edges is subdivided and ℓ additional copies of each of that subdivisions are introduced, that is, if $\{v, w\}$ is one of these additional edges, then the vertices $x_1^{vw}, \ldots, x_{\ell+1}^{vw}$ are added and edge $\{v, w\}$ is replaced by the edges $\{\{v, x_1^{vw}\}, \ldots, \{v, x_{\ell+1}^{vw}\}, \{x_1^{vw}, w\}, \ldots, \{x_{\ell+1}^{vw}, w\}\}$. In the following, we denote these paths by *copy paths*. The resulting graph contains all minimal s-t separators of size at most ℓ in G and has treewidth upper-bounded by $h(\ell)$ for some function h only depending on ℓ.

▶ **Theorem 4** (Treewidth reduction [17, Theorem 2.15]). *Let G be a graph, $T \subseteq V(G)$, and let ℓ be an integer. Let C be the set of all vertices of G participating in a minimal s-t separator of size at most ℓ for some $s, t \in T$. For every fixed ℓ and $|T|$, there is a linear-time algorithm that computes a graph G^* having the following properties:*

1. *$C \cup T \subseteq V(G^*)$.*
2. *For every $s, t \in T$, a set $L \subseteq V(G^*)$ with $|L| \leq \ell$ is a minimal s-t separator of G^* if and only if $L \subseteq C \cup T$ and L is a minimal s-t separator of G.*
3. *The treewidth of G^* is at most $h(\ell, |T|)$ for some function h.*
4. *$G^*[C \cup T]$ is isomorphic to $G[C \cup T]$.*

For finding a p-routing we are interested in minimal s-t cuts of size smaller than p in G. The treewidth reduction technique guarantees to preserve minimal s-t separators of a specific size, but does not guarantee to preserve minimal s-t cuts of a specific size. Thus, we need to modify our graph G in such a way that each minimal s-t cut in G corresponds to a minimal s-t separator in the modified graph. We modify graph G in the following way.

▶ **Step 1.** *Subdivide each edge in $E(G)$, that is, for each edge $e = \{v, w\}$ in $E(G)$ add a new vertex x_e and replace edge e by edge $\{v, x_e\}$ and edge $\{x_e, w\}$. We say that vertex x_e as well as edges $\{v, x_e\}$ and $\{x_e, w\}$* correspond *to edge e. Let $V_E := \{x_e \mid e \in E\}$ and E' be the edge set replacing the edges in E. Then $H := (V \cup V_E, E')$ is the resulting graph.*

Note that each edge in H is incident with exactly one vertex in V_E and one vertex in V. Thus, no two vertices in V_E and no two vertices in V are neighbors. Moreover, note that each vertex in V_E has degree exactly two. It holds that $|V \cup V_E| = |V| + |E|$ and $|E'| = 2 \cdot |E|$.

Recall that we are interested in s-t cuts in G. By our modification from Step 1 of G to H, for each edge in G there is a corresponding vertex in V_E in H. One can show that there is a one-to-one correspondence between s-t cuts in G and those s-t separators in H that contain only vertices in V_E. Moreover, the following lemma holds.

▶ **Lemma 5.** *Every minimal s-t cut in G corresponds to a minimal s-t separator in H.*

Next, we show that every vertex in the neighborhood of each minimal s-t separator containing only vertices in V_E belongs to a minimal s-t separator.

▶ **Lemma 6.** *Let $W \subseteq V_E \subseteq V(H)$ be the set of vertices corresponding to a minimal s-t cut of size at most $\ell \in \mathbb{N}$ in G. Then, each vertex in $N_H[W]$ is part of a minimal s-t separator of size at most ℓ in H.*

We obtained graph H from graph G by applying Step 1. By Theorem 5, we know that each minimal s-t cut in G corresponds to a minimal s-t separator in H. Moreover, by Theorem 6, if we consider a minimal s-t cut of size smaller than p in G, then, for each neighbor of the vertex set in H corresponding to the minimal s-t cut in G, there exists a minimal s-t separator of size smaller than p in H that contains that neighbor. As the next step (cf. Figure 1) we apply the treewidth reduction technique [17] to graph H.

▶ **Step 2.** *Apply the treewidth reduction (Theorem 4) to graph H with $T = \{s, t\}$ and $p - 1$ as upper bound for the size of the minimal s-t separators. Denote the resulting graph by H^*.*

Let $V_E^* := \{v \in V(H^*) \mid v \in V_E\}$. Graph H^* contains all minimal s-t separators of size at most $p - 1$ in H. By Theorem 5, every minimal s-t cut of size at most $p - 1$ in G corresponds to a minimal s-t separator of size at most $p - 1$ in H and thus, by Theorem 4, to a minimal s-t separator of size at most $p - 1$ in H^*. By Theorem 6, the neighborhood of

each vertex in H corresponding to a vertex in V_E^* is contained in the vertex set $V(H^*)$. As a consequence, we can reconstruct each edge in graph G that appears in a minimal s-t cut of size at most $p-1$ in G as an edge in the graph H^*. As our next step (cf. Figure 1), we contract for each vertex in V_E^* an incident edge in graph H^*. We remark that if x^{vw} is a vertex in V_E^*, then the only edges incident with vertex x^{vw} are $\{v, x^{vw}\}$ and $\{x^{vw}, w\}$. In addition, the vertices v and w are the only neighbors of x^{vw} in graph H and in graph H^*.

▶ **Step 3.** *Contract for each vertex in V_E^* exactly one incident edge in H^* to obtain the graph G^*. In other words, undo the subdivision applied on G to obtain H.*

We remark that $\mathsf{tw}(G^*) \leq \mathsf{tw}(H^*)$, since edge contraction does not increase the treewidth of a graph [22].

Let $e = \{v, w\} \in E(G)$ be an edge in G and $x_e \in V_E \subseteq V(H)$ be the corresponding vertex in H. Then $\{v, x_e\}$ and $\{x_e, w\}$ are the incident edges of x_e in H. If $x_e \in V(H^*)$, then one of the incident edges $\{v, x_e\}$ and $\{x_e, w\}$ with vertex x_e is contracted and yields edge $\{v, w\} \in E(G^*)$. We say that the edges $\{v, w\} \in E(G)$ and $\{v, w\} \in E(G^*)$ correspond one-to-one, and, for example, we write $\{v, w\} \in E(G) \cap E(G^*)$.

Considering the graphs G and G^*, we remark that one can show that, given an s-t path in the one graph, one can find an s-t path in the other graph using a common set of edges in $E(G) \cap E(G^*)$. The next lemma states that each minimal s-t cut of size smaller than p in one of the graphs G and G^* is also a minimal s-t cut of size smaller than p in the other graph.

▶ **Lemma 7.** *Let $C \subseteq E(G) \cap E(G^*)$. Edge set C is a minimal s-t cut in G of size smaller than p if and only if C is a minimal s-t cut in G^* of size smaller than p.*

Recalling Theorem 3, we know that if an instance of MSE is a yes-instance, then we can find k edges such that the k edges form a solution for the instance and each of the k edges is part of a minimal s-t cut of size smaller than p in G. By Theorem 7, the graphs G and G^* have the same set of minimal s-t cuts of size smaller than p. Combining Theorem 3 and Theorem 7 leads to the following lemma.

▶ **Lemma 8.** *(G^*, s, t, p, k) is a yes-instance of* MSE *if and only if (G, s, t, p, k) is a yes-instance of* MSE*.*

By Theorem 8, we know that the instances (G^*, s, t, p, k) and (G, s, t, p, k) are equivalent for MSE. By our construction, we know that the treewidth of G^* is upper-bounded by a function only depending on the number p of routes. In addition, we know that Minimum Shared Edges is fixed-parameter tractable with respect to the number p of routes and an upper bound on the treewidth of the input graph. Thus, we are ready to prove Theorem 2.

Proof of Theorem 2. First we modify our graph $G = (V, E)$ by applying Steps 1 to 3. Let H, H^*, and G^* be the according graphs. By Theorem 4, the treewidth of H^* is upper-bounded by $h(p)$ for some function h. Since edge contractions do not increase the treewidth of a graph [22], it follows that $\mathsf{tw}(G^*) \leq \mathsf{tw}(H^*)$. By Theorem 8, the instances (G^*, s, t, p, k) and (G, s, t, p, k) are equivalent for MSE.

We know from Theorem 1 that MSE(p, ω) is fixed-parameter tractable when parameterized by the number p of routes and by an upper bound ω on the treewidth of the input graph. Since function h only depends on p and $h(p)$ is upper-bounding the treewidth of graph G^*, we can solve instance (G^*, s, t, p, k) in $f(p) \cdot O(|V(G^*)|)$ time, where f is a computable function only depending on parameter p. Since $|V(G^*)| \leq |V(G)| + p \cdot |E(G)| \leq p \cdot |G|$ and the instances (G^*, s, t, p, k) and (G, s, t, p, k) are equivalent for MSE, we can decide instance (G, s, t, p, k) in $f(p) \cdot p \cdot O(|G|)$ time, that is, in FPT-time. ◀

Using the dynamic program from Section 3 the running time of the above algorithm amounts to $O(p^2 \cdot (h(p)+4)^{3\cdot p\cdot(h(p)+3)+3} \cdot |G|)$. Using the bound $h(p) \leq 2^{O(p^2)}$ [17], we obtain a running time of $2^{p^3\cdot 2^{O(p^2)}} \cdot (n+m)$.

5 No Polynomial Kernel for the Parameter Number of Routes

In the previous section, we showed that Minimum Shared Edges is fixed-parameter tractable with respect to the number p of routes. It is well known that a problem is fixed-parameter tractable if and only if it admits a problem kernel. Of particular interest is the minimal possible size of a problem kernel. Accordingly, in this section we present the following lower bound.

▶ **Theorem 9.** Minimum Shared Edges *does not admit a polynomial-size problem kernel with respect to the number p of routes, unless NP $\subseteq$ coNP/poly.*

We prove Theorem 9 via an *OR-cross-composition* [4], that is, given ℓ instances of an NP-hard problem Q, all contained in one equivalence class of a polynomial-time computable relation $\mathcal{R}$ of our choosing, we compute in polynomial-time an instance (G, s, t, p, k) of MSE such that

(i) p is bounded by a polynomial function of the size of the largest input instance (*boundedness*), and

(ii) (G, s, t, p, k) is a yes-instance if and only if one of the input instances is a yes-instance (*correctness*).

If this is possible, then MSE does not admit a polynomial-size problem kernel with respect to p unless NP $\subseteq$ coNP/poly [4].

It is tempting to use MSE itself as the problem Q, to assume that each of the instances asks for the same number of routes and same number of shared edges by virtue of $\mathcal{R}$, and to OR-cross-compose by simply gluing the graphs in a chain-like fashion on sinks and sources. This fulfills the boundedness constraint, but not necessarily the correctness constraint, since the instances can "share" shared edges between them. Hence, we use the following problem as the problem Q instead.

Almost Minimum Shared Edges (AMSE)
Input: An undirected graph G, two distinct vertices $s, t \in V(G)$, and two integers $p, k \in \mathbb{N}$ such that G has a (p, s, t)-routing with at most $k+1$ shared edges.
Question: Is there a (p, s, t)-routing in G with at most k shared edges?

▶ **Proposition 10.** Almost Minimum Shared Edges *is NP-hard.*

Proposition 10 can be proven via a reduction from MSE to AMSE that introduces an additional path of length $k+1$ connecting s and t.

If we OR-cross-compose from AMSE instead, we know that if the resulting instance has a p-routing with $\ell(k+1)-1$ shared edges, then without loss of generality each of the original instances contributes at most $k+1$ shared edges. This means that at least one of the original instances is a yes-instance, giving the correctness of the OR-cross-composition.

6 W[1]-hardness with Respect to Treewidth

In this section, we present the following result.

▶ **Theorem 11.** Minimum Shared Edges *is W[1]-hard when parameterized by treewidth and the number k of shared edges combined.*

To prove Theorem 11, we give a parameterized reduction from the following problem. Herein, $\dot\cup$ denotes the disjoint union of sets.

Multicolored Clique (MCC)
Input: An undirected, k-partite graph $G = (V = V_1 \dot\cup \ldots \dot\cup V_k, E)$ with $k \in \mathbb{N}$.
Question: Is there a set $C \subseteq V$ of vertices such that $G[C]$ is a k-clique in G?

MCC is W[1]-complete when parameterized by k [11]. In the remainder of the section (G, k) is an arbitrary but fixed instance of MCC. We denote $|V_i| =: n_i$ and $V_i =: \{v_1^i, \ldots, v_{n_i}^i\}$ for all $i \in [k]$. We also say that G has the *color classes* $1, \ldots, k$, where each color class i is represented by the vertices in V_i. We write $E_{i,j} := \{\{v, w\} \in E \mid v \in V_i, w \in V_j\}$ for the edges connecting vertices in V_i and V_j, $i, j \in [k]$.

The reduction is based on the following idea. The routes we are to allocate will be split evenly into contingents of routes for each color class by a simple gadget. For each of the color classes, we introduce a selection gadget, that contains vertices (*outputs*) that correspond to the vertices in the MCC instance. Each selection gadget will route almost all the routes in its contingent to exactly one of its outputs. The outputs will then disperse $(k-1)$-times a number of routes corresponding to the ID of the vertex that this output represents. In this way, the selection gadgets represent a choice of vertices, one for each color class. In order to verify that the choice represents a clique, we introduce validation gadgets, corresponding to the pairs of color classes. They will receive the routes from the outputs of the selection gadgets, that is, the "input" of the validation gadgets is a sum of two IDs. They induce a small number of shared edges only if the vertices according to the number of routes are connected. In order to achieve this, we ensure that the sum of two IDs uniquely identifies the vertices. We achieve this by using Sidon sets.

Vertex IDs based on Sidon sets. A *Sidon set* is a set $S \subseteq \mathbb{N}$ that fulfills that for each $i, j, k, \ell \in S$ holds that if $i + k = j + \ell$ then $\{i, k\} = \{j, \ell\}$. That is, the sum of any two distinct elements in S is unique. A Sidon set S with $\max_{i \in S} i \in O(|S|^3)$ can be constructed on $O(|S|)$ time [9, page 42]. As mentioned, we use a Sidon set to distinguish numbers of routes corresponding to vertices. For this purpose, we fix a Sidon set S with $|S| = |V|$ and assign to each vertex $v \in V$ an *ID* $g(v) \in S$ where g is a bijection. For technical reasons, we need the following additional properties of g (and S):

(i) $g(v) \geq n^3$ for all $v \in V$,
(ii) $|g(v) - g(w)| \geq n^3$ for all $v, w \in V$, $v \neq w$, and
(iii) $|(g(v) + g(w)) - (g(x) + g(y))| \geq n^3$ for all $v, w, x, y \in V$, $v \neq w$, $y \notin \{v, w, x\}$.

Clearly, by adding one to each integer in the Sidon set S and then multiplying each integer by n^3 we obtain a Sidon set and a mapping g that fulfill all of the above properties simultaneously.

To enforce that only adjacent vertices are chosen in the selection gadgets, a part in a validation gadget that represents an edge must have the property that, if many routes are routed through it, then the number of routes corresponds to precisely the sum of IDs of the endpoints of the edge that is represented by this part. To do this, we have to enforce both upper and lower bounds on the sum of IDs. Upper bounds will be enforced by long parallel paths; for lower bounds, we use the notion of "complement" of an ID. For this, we define $\overline{g(v)} := M - g(v)$ for all $v \in V$, where $M := n^3 + \max_{v \in V} g(v)$. Note that $g(v) + g(w) < g(x) + g(y)$ if and only if $\overline{g(v)} + \overline{g(w)} > \overline{g(x)} + \overline{g(y)}$ for $v, w, x, y \in V$.

Construction

In the following, we describe the construction of the instance (G', s, t, p, k') of MSE, given instance (G, k) of MCC. Initially, G' consists only of the two vertices s and t, the source and the sink vertex, respectively. We describe the gadgets we use and their interconnections, which will fully describe the construction of G'. As mentioned, our gadgetry consists of two gadget types, selection gadgets on the one hand and validation gadgets on the other hand.

Before we proceed, we fix the following notation. An *m-chain* is a P_{m+1}, i.e., a path of length m. A set of ℓ m-chains with common endpoints we call an *(ℓ, m)-bundle*. A *(q, ℓ, m)-feather* is obtained by identifying one endpoint of an (ℓ, m)-bundle with one endpoint of a q-chain. In the following, by attaching a chain, bundle, or feather H to a vertex v, we mean to identify v with an endpoint of H.

We set the number of paths $p = \left(|E| - \binom{k}{2}\right) + k \cdot ((k-1) \cdot M + 1) + n$ and the number of shared edges $k' = k \cdot k^{10} + k \cdot (k + 2(k-1)) \cdot k^5 + \binom{k}{2} \cdot 3k$.

Selection gadgets. For each color class $i \in [k]$ in the instance (G, k), we construct a selection gadget $\boxed{i}$ that selects exactly one vertex of V_i as follows.

We introduce vertex c_i corresponding to color class i in (G, k). We connect s with c_i via a $((k-1) \cdot M + n_i + 1, k' + 1)$-bundle. Each of the chains in the bundle will be in exactly one route later. We introduce the vertices $x^i_1, \ldots, x^i_{n_i}$ in G', corresponding to the vertices $v^i_1, \ldots, v^i_{n_i} \in V_i$, and we connect c_i to each of them by a k^{10}-chain. These vertices serve as hubs for the routes later; only one of them will carry almost all routes in any solution, representing the choice of a vertex into the clique.

In order to relay this choice to all the validation gadgets, we do the following. First, we attach a k-chain to each vertex x^i_j, $1 \leq j \leq n_i$. Let $x^i_{j,1}, \ldots, x^i_{j,k}$ denote the vertices on the chain attached to x^i_j, indexed by the distance on the chain to vertex x^i_j; each vertex except $x^i_{j,k}$ will make its own connection to the validation gadgets. We connect each $x^i_{j,\ell}$, $\ell \in [k-1]$, with the vertex $c_i c_{\ell'}$ in the validation gadget $\boxed{i, \ell'}$ (introduced below), where $\ell' = \ell$ if $\ell < i$ and $\ell' = \ell + 1$ otherwise. The connection is made by attaching a $(k^5, g(v^i_j), k' + 1)$-feather to $x^i_{j,\ell}$ and $c_i c_{\ell'}$. Furthermore, to relay also the complement IDs, we connect each $x^i_{j,\ell}$, $\ell \in [k-1]$, with the vertex $\overline{c_i c_{\ell'}}$ by attaching a $(k^5, \overline{g(v^i_j)}, k' + 1)$-feather to them. We k^5-subdivide each edge on the k-chain we attached to x^i_j, that is, we replace each edge by a k^5-chain. We apply this to all edges on the path $x^i_1, \ldots, x^i_{n_i}$. This will ensure that in each color class, only the "ID relay vertices" $x^i_{j,\ell}$ corresponding to one ID will carry more than one route. Note that the only differences between the ID relay vertices are the second entries of the feathers, which depend on the corresponding values of the Sidon set. Finally, we connect vertex $x^i_{j,k}$ to t via a $(2, k' + 1)$-bundle; this vertex ensures that each k^5-chain between two vertices $x^i_{j,\ell}$ corresponding to the chosen ID is shared.

Validation gadgets. We need to check that the chosen vertices are adjacent using only their IDs. For this we encode the sums of IDs corresponding to two adjacent vertices into a bundle which has to be passed by the routes relayed from the selection gadgets. The budget will not allow to share any of the paths in this bundle. In this way, any sum of IDs has to be below a certain threshold. To get a lower bound, we also introduce bundles for sums of complement IDs of adjacent vertices. Finally, we ensure that an "ID" bundle and its "complement ID" bundle can be used simultaneously, only if they correspond to the same pair of vertices.

We now describe the construction of a validation gadget $\boxed{i,j}$, $i,j \in [k]$, $i < j$. We introduce exactly two vertices c_ic_j and $\overline{c_ic_j}$ (recall that these vertices already appeared in the description of the selection gadgets). We introduce a vertex for each edge between V_i and V_j, that is, if $\{v_y^i, v_z^j\} \in E_{i,j}$, then we introduce the vertex $x_y^i x_z^j$ in G'. We connect each $x_y^i x_z^j$ to c_ic_j by attaching a $(k, g(v_y^i) + g(v_z^j), k'+1)$-feather, we connect $x_y^i x_z^j$ to $\overline{c_ic_j}$ by attaching a $(k, \overline{g(v_y^i)} + \overline{g(v_z^j)}, k'+1)$-feather, and we connect $x_y^i x_z^j$ to the sink vertex t by attaching a k-chain. Only one of the connections to the sink will carry more than one route; hence, it will be possible to use only one pair of complementary bundles (corresponding to a pair of adjacent vertices).

For technical reasons, we need that each pair of bundles carries at least one route; this is achieved by also connecting s with c_ic_j via an $(|E_{i,j}| - 1, k'+1)$-bundle.

The correctness proof is deferred to the full version.

Upper-Bound on the Treewidth

To construct a tree decomposition of small width, we start out with a single bag A, where $A := \{s\} \cup \{t\} \cup \{c_i \mid i \in [k]\} \cup \{c_ic_j \mid 1 \leq i < j \leq k\} \cup \{\overline{c_ic_j} \mid 1 \leq i < j \leq k\}$. Note that $|A| = 2 + k + 2\binom{k}{2}$. Since all gadgets are interconnected via only vertices from the set A, in order to construct a tree decomposition for G', we can build a tree decomposition $\mathbb{T}'$ of each gadget separately, then add A to each of its bags, and then attach $\mathbb{T}'$ to the bag A we started with. Observe that each chain, bundle, and feather is a series-parallel graph. Since each gadget allows a tree-like structure where each edge corresponds to a series-parallel graph and each leaf is contained in A, we can find a tree decomposition of width at most 4 for each gadget. Hence, the treewidth of the graph G' as constructed above is upper-bounded by $2\binom{k}{2} + k + 2 + 4$.

7 Conclusion

Minimum Shared Edges (MSE) is a fundamental NP-hard network routing problem. We focused on exact solutions for the case of undirected, general graphs and provided several classification results concerning the parameterized complexity of MSE.

It is fair to say that our fixed-parameter tractability results (based on tree decompositions and the treewidth reduction technique [17]) are still far from practical relevance. Our studies indicate, however, that MSE is a natural candidate for performing a wider multivariate complexity analysis [12, 19] as well as studying restrictions to special graph classes. For instance, there is a simple search tree algorithm solving MSE in $O((p-1)^k \cdot (m+n)^2)$ time which might be useful in some applications [14]. Moreover, it can be shown that on unbounded undirected grids (without holes), due to combinatorial arguments, MSE can be decided in constant time after reading the input [14]. On the contrary, ongoing work indicates that MSE remains NP-hard when restricted to planar graphs (which might be of particular relevance when studying street networks). NP-hardness also prevails in case of graphs with maximum degree five [14].

In the known (pseudo) polynomial-time algorithms for graphs of bounded treewidth the exponents in the running time depend exponentially on the treewidth [1, 2]. It would be interesting to know whether a polynomial dependence is achievable.

A further line of future work is to study closely related problems and natural variants of MSE. For instance, can the positive results be transferred to the more general Minimum Vulnerability problem [2] (see the introductory section)? There are also some preliminary

investigations concerning the problem Short Minimum Shared Edges (with an additional upper bound on the maximum length of a route) [14]. Finally, it is natural to study "time-sharing" aspects for the shared edges, yielding a further natural variant of MSE.

References

1 Yusuke Aoki, Bjarni V. Halldórsson, Magnús M. Halldórsson, Takehiro Ito, Christian Konrad, and Xiao Zhou. The minimum vulnerability problem on graphs. In *Proc. 8th International Conference on Combinatorial Optimization and Applications (COCOA'14)*, volume 8881 of *LNCS*, pages 299–313. Springer, 2014.

2 Sepehr Assadi, Ehsan Emamjomeh-Zadeh, Ashkan Norouzi-Fard, Sadra Yazdanbod, and Hamid Zarrabi-Zadeh. The minimum vulnerability problem. In *Proc. 23rd International Symposium on Algorithms and Computation (ISAAC'12)*, volume 7676 of *LNCS*, pages 382–391. Springer, 2012.

3 René van Bevern, Andreas Emil Feldmann, Manuel Sorge, and Ondřej Suchý. On the parameterized complexity of computing balanced partitions in graphs. *Theory of Computing Systems*, 57(1):1–35, 2015.

4 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

5 Rajesh Chitnis, László Egri, and Dániel Marx. List H-coloring a graph by removing few vertices. In *Proc. 21st European Symposion on Algorithms (ESA '13)*, volume 8125 of *LNCS*, pages 313–324. Springer, 2013.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

7 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proc. IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS'11)*, pages 150–159, 2011.

8 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.

9 Apostolos Dimitromanolakis. Analysis of the Golomb ruler and the Sidon set problems, and determination of large, near-optimal Golomb rulers. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, June 2002. `http://www.cs.toronto.edu/~apostol/golomb/`.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

11 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

12 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

14 Till Fluschnik. The parameterized complexity of finding paths with shared edges. Master thesis, Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, 2015. `http://fpt.akt.tu-berlin.de/publications/theses/MA-till-fluschnik.pdf`.

15 Gregory Gutin, Mark Jones, and Bin Sheng. Parameterized complexity of the k-arc Chinese postman problem. In *Proc. 22nd European Symposium on Algorithms (ESA '14)*, volume 8737 of *LNCS*, pages 530–541. Springer, 2014.

16 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

17 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013.

18 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

19 Rolf Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proc. 27th International Symposium on Theoretical Aspects of Computer Science (STACS '10)*, volume 5 of *LIPIcs*, pages 17–32. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

20 Masoud T. Omran, Jörg-Rüdiger Sack, and Hamid Zarrabi-Zadeh. Finding paths with minimum shared edges. *Journal of Combinatorial Optimization*, 26(4):709–722, 2013.

21 Andrzej Pelc. Fault-tolerant broadcasting and gossiping in communication networks. *Networks*, 28(3):143–156, 1996.

22 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

23 Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, 2 edition, 2000.

24 Zhi-Qian Ye, Yi-Ming Li, Hui-Qiang Lu, and Xiao Zhou. Finding paths with minimum shared edges in graphs with bounded treewidths. In *Proc. Frontiers of Computer Science (FCS'13)*, pages 40–46, 2013.

Control Improvisation

Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel

University of California, Berkeley, CA, US
{dfremont,donze,sseshia}@berkeley.edu

Abstract

We formalize and analyze a new automata-theoretic problem termed *control improvisation*. Given an automaton, the problem is to produce an *improviser*, a probabilistic algorithm that randomly generates words in its language, subject to two additional constraints: the satisfaction of an *admissibility* predicate, and the exhibition of a specified amount of randomness. Control improvisation has multiple applications, including, for example, generating musical improvisations that satisfy rhythmic and melodic constraints, where admissibility is determined by some bounded divergence from a reference melody. We analyze the complexity of the control improvisation problem, giving cases where it is efficiently solvable and cases where it is #P-hard or undecidable. We also show how symbolic techniques based on Boolean satisfiability (SAT) solvers can be used to approximately solve some of the intractable cases.

1998 ACM Subject Classification F.4.3 Formal Languages, G.3 Probability and Statistics, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases finite automata, random sampling, Boolean satisfiability, testing, computational music, control theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.463

1 Introduction

We introduce and formally characterize a new automata-theoretic problem termed *control improvisation*. Given an automaton, the problem is to produce an *improviser*, a probabilistic algorithm that randomly generates words in the language of the automaton, subject to two additional constraints: each generated word must satisfy an *admissibility* predicate, and the improviser must exhibit a specified amount of randomness.

The original motivation for this problem arose from a topic known as *machine improvisation of music* [19]. Here, the goal is to create algorithms which can generate variations of a reference melody like those commonly improvised by human performers, for example in jazz. Such an algorithm should have three key properties. First, the melodies it generates should conform to rhythmic and melodic constraints typifying the music style (e.g. in jazz, the melodies should follow the harmonic conventions of that genre). Second, the algorithm should be sufficiently randomized that running it several times produces a variety of different improvisations. Finally, the generated melodies should be actual variations on the reference melody, neither reproducing it exactly nor being so different as to be unrecognizable. In previous work [10], we identified these properties in an initial definition of the control improvisation problem, and applied it to the generation of monophonic (solo) melodies over a given jazz song harmonization[1].

[1] Examples of improvised melodies can be found at the following URL: http://www.eecs.berkeley.edu/~donze/impro_page.html.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 463–474

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

These three properties of a generation algorithm are not specific to music. Consider *black-box fuzz testing* [20], which produces many inputs to a program hoping to trigger a bug. Often, constraints are imposed on the generated inputs, e.g. in *generative* fuzz testing approaches which enforce an appropriate format so that the input is not rejected immediately by a parser. Also common are *mutational* approaches which guide the generation process with a set of real-world seed inputs, generating only inputs which are variations of those in the set. And of course, fuzzers use randomness to ensure that a variety of inputs are tried. Thus we see that the inputs generated in fuzz testing have the same general requirements as music improvisations: satisfying a set of constraints, being appropriately similar/dissimilar to a reference, and being sufficiently diverse.

We propose control improvisation as a precisely-defined theoretical problem capturing these requirements, which are common not just to the two examples above but to many other generation problems. Potential applications also include home automation mimicking typical occupant behavior (e.g., randomized lighting control obeying time-of-day constraints and limits on energy usage [16]) and randomized variants of the supervisory control problem [5], where a controller keeps the behavior of a system within a safe operating region (the language of an automaton) while adding diversity to its behavior via randomness. A typical example of the latter is surveillance: the path of a patrolling robot should satisfy various constraints (e.g. not running into obstacles) and be similar to a predefined route, but incorporate some randomness so that its location is not too predictable [15].

Our focus, in this paper, is on the *theoretical characterization of control improvisation*. Specifically, we give a precise theoretical definition and a rigorous characterization of the complexity of the control improvisation problem under various conditions on the inputs to the problem. While the problem is distinct from any other we have encountered in the literature, our methods are closely connected to prior work on random sampling from the languages of automata and grammars [13, 9, 14], and sampling from the satisfying assignments of a Boolean formula [6]. Probabilistic programming techniques [12] could be used for sampling under constraints, but the present methods cannot be used to construct improvisers meeting our definition.

In summary, this paper makes the following novel contributions:

- Formal definitions of the notions of control improvisation (CI) and a polynomial-time improvisation scheme (Sec. 2);
- A theoretical characterization of the conditions under which improvisers exist (Sec. 3);
- A polynomial-time improvisation scheme for a practical class of CI instances, involving finite-memory admissibility predicates (Sec. 4);
- #P-hardness and undecidability results for more general classes of the problem (Sec. 5);
- A symbolic approach based on Boolean satisfiability (SAT) solving that is useful in the case when the automata are finite-state but too large to represent explicitly (Sec. 6).

We conclude in Sec. 7 with a synopsis of results and directions for future work. For lack of space, we include only selected proofs and proof sketches in the main body of the paper; complete details may be found in the Appendix of the full version [11].

2 Background and Problem Definition

In this section, we first provide some background on a previous automata-theoretic method for music improvisation based on a data structure called the *factor oracle*. We then provide a formal definition of the control improvisation problem while explaining the choices made in this definition.

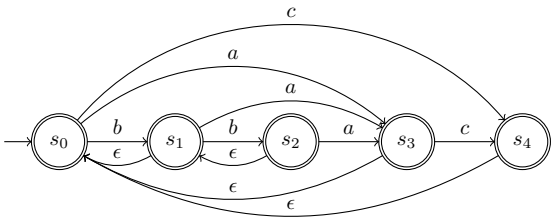


Figure 1 Factor oracle constructed from the word $w_{\text{ref}} = bbac$.

2.1 Factor Oracles

An effective and practical approach to machine improvisation of music (used for example in the prominent OMax system [1]) is based on a data structure called the factor oracle [2, 7]. Given a word w_{ref} of length N that is a symbolic encoding of a reference melody, a factor oracle F is an automaton constructed from w_{ref} with the following key properties: F has $N+1$ states, all accepting, chained linearly with direct transitions labelled with the letters in w_{ref}, and with potentially additional forward and backward transitions. Figure 1 depicts F for $w_{\text{ref}} = bbac$. A word w accepted by F consists of concatenated "factors" of w_{ref}, and its dissimilarity with w_{ref} is correlated with the number of non-direct transitions. By assigning a small probability α to non-direct transitions, F becomes a generative Markov model with tunable "divergence" from w_{ref}. In order to impose more musical structure on the generated words, our previous work [10] additionally requires that improvisations satisfy rules encoded as deterministic finite automata, by taking the product of the generative Markov model and the DFAs. While this approach is heuristic and lacks any formal guarantees, it has the basic elements common to machine improvisation schemes: (i) it involves randomly generating strings from a formal language typically encoded as an automaton, (ii) it enforces diversity in the generated strings, and (iii) it includes a requirement on which strings are admissible based on their divergence from a reference string. The definition we propose below captures these elements in a rigorous theoretical manner, suitable for further analysis. In Sec. 4, we revisit the factor oracle, sketching how the notion of divergence from w_{ref} that it represents can be encoded in our formalism.

2.2 Problem Definition

We abbreviate deterministic and nondeterministic finite automata as DFAs and NFAs respectively. We use the standard definition of probabilistic finite automata from [18], where a string is accepted iff it causes the automaton to reach an accepting state with probability greater than a specified *cut-point* $p \in [0, 1)$. We call a probabilistic finite automaton, together with a choice of cut-point so that its language is definite, a PFA. We write $\Pr[f(X) \mid X \leftarrow D]$ for the probability of event $f(X)$ given that the random variable X is drawn from the distribution D.

▶ **Definition 2.1.** An *improvisation automaton* is a finite automaton (DFA, NFA, or PFA) $\mathcal{I}$ over a finite alphabet Σ. An *improvisation* is any word $w \in L(\mathcal{I})$, and $I = L(\mathcal{I})$ is the set of all improvisations.

▶ **Definition 2.2.** An *admissibility predicate* is a computable predicate $\alpha : \Sigma^* \to \{0, 1\}$. An improvisation $w \in I$ is *admissible* if $\alpha(w) = 1$. We write A for the set of all admissible improvisations.

Running Example. Our concepts will be illustrated with a simple example. Our aim is to produce variations of the binary string $s = 001$ of length 3, subject to the constraint that there cannot be two consecutive 1s. So $\Sigma = \{0, 1\}$, and $\mathcal{I}$ is a DFA which accepts all length-3 strings that do not have two 1s in a row. To ensure that our variations are similar to s, we let our admissibility predicate $\alpha(w)$ be 1 if the Hamming distance between w and s is at most 1, and 0 otherwise. Then the improvisations are the strings 000, 001, 010, 100, and 101, of which 000, 001, and 101 are admissible.

Intuitively, an improviser samples from the set of improvisations according to some distribution. But what requirements must one impose on this distribution? Since we want a variety of improvisations, we require that each one is generated with probability at most some bound ρ. By choosing a small value of ρ we can thus ensure that many different improvisations can be generated, and that no single one is output too frequently. Other constraints are possible, e.g. requiring that every improvisation have nonzero probability, but we view this as too restrictive: if there are a large number of possible improvisations, it should be acceptable for an improviser to generate many but not all of them. Another possibility would be to ensure variety by imposing some minimum distance between the improvisations. This could be reasonable in a setting (such as music) where there is a natural metric on the space of improvisations, but we choose to keep our setting general and not assume such a metric. Finally, we require our generated improvisation to be admissible with probability at least $1 - \epsilon$ for some specified ϵ. When the admissibility predicate encodes a notion of similarity to a reference string, for example, this allows us to require that our improvisations usually be similar to the reference. Combining these requirements, we obtain our definitions of an acceptable distribution over improvisations and thus of an improviser:

▶ **Definition 2.3.** Given $\mathcal{C} = (\mathcal{I}, \alpha, \epsilon, \rho)$ with $\mathcal{I}$ and α as in Definitions 2.1 and 2.2, $\epsilon \in [0, 1] \cap \mathbb{Q}$ an error probability, and $\rho \in (0, 1] \cap \mathbb{Q}$ a probability bound, a distribution $D : \Sigma^* \to [0, 1]$ with support S is an (ϵ, ρ)-*improvising distribution* if:

- $S \subseteq I$
- $\forall w \in S, D(w) \leq \rho$
- $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$

If there is an (ϵ, ρ)-improvising distribution, we say that $\mathcal{C}$ is (ϵ, ρ)-*feasible* (or simply *feasible*). An (ϵ, ρ)-*improviser* (or simply *improviser*) for a feasible $\mathcal{C}$ is an expected finite-time probabilistic algorithm generating strings in Σ^* whose output distribution (on empty input) is an (ϵ, ρ)-improvising distribution.

To summarize, if $\mathcal{C} = (\mathcal{I}, \alpha, \epsilon, \rho)$ is feasible, there exists a distribution satisfying the requirements in Definition 2.3, and an improviser is a probabilistic algorithm for sampling from one.

Running Example. For our running example, $\mathcal{C} = (\mathcal{I}, \alpha, 0, 1/4)$ is not feasible since $\epsilon = 0$ means we can only generate admissible improvisations, and since there are only 3 of those we cannot possibly give them all probability at most $1/4$. Increasing ρ to $1/3$ would make $\mathcal{C}$ feasible. Increasing ϵ to $1/4$ would also work, allowing us to return an inadmissible improvisation $1/4$ of the time: an algorithm uniformly sampling from $\{000, 001, 101, 100\}$ would be an improviser for $(\mathcal{I}, \alpha, 1/4, 1/4)$.

▶ **Definition 2.4.** Given $\mathcal{C} = (\mathcal{I}, \alpha, \epsilon, \rho)$, the *control improvisation (CI)* problem is to decide whether $\mathcal{C}$ is feasible, and if so to generate an improviser for $\mathcal{C}$.

Ideally, we would like an efficient algorithm to solve the CI problem. Furthermore, the improvisers our algorithm produces should themselves be efficient, in the sense that their runtimes are polynomial in the size of the original CI instance. This leads to our last definition:

▶ **Definition 2.5.** A *polynomial-time improvisation scheme* for a class $\mathcal{P}$ of CI instances is a polynomial-time algorithm S with the following properties:

- for any $\mathcal{C} \in \mathcal{P}$, if $\mathcal{C}$ is feasible then $S(\mathcal{C})$ is an improviser for $\mathcal{C}$, and otherwise $S(\mathcal{C}) = \bot$
- there is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that if $G = S(\mathcal{C}) \neq \bot$, then G has expected runtime at most $p(|\mathcal{C}|)$.

A polynomial-time improvisation scheme for a class of CI instances is an efficient, uniform way to solve the control improvisation problem for that class. In Sections 4 and 5 we will investigate which classes have such improvisation schemes.

3 Existence of Improvisers

It turns out that the feasibility of an improvisation problem is completely determined by the sizes of I and A:

▶ **Theorem 3.1.** *For any $\mathcal{C} = (\mathcal{I}, \alpha, \epsilon, \rho)$, the following are equivalent:*

(a) *$\mathcal{C}$ is feasible.*
(b) *$|I| \geq 1/\rho$ and $|A| \geq (1-\epsilon)/\rho$.*
(c) *There is an improviser for $\mathcal{C}$.*

Proof. **(a)⇒(b):** Suppose D is an (ϵ, ρ)-improvising distribution with support S. Then $\rho|S| = \sum_{w \in S} \rho \geq \sum_{w \in S} D(w) = 1$, so $|I| \geq |S| \geq 1/\rho$. We also have $\rho|S \cap A| = \sum_{w \in S \cap A} \rho \geq \sum_{w \in S \cap A} D(w) = \Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$, so $|A| \geq |S \cap A| \geq (1-\epsilon)/\rho$.
(b)⇒(c): Defining $N = \lceil (1-\epsilon)/\rho \rceil$, we have $|A| \geq N$. If $N \geq 1/\rho$, then there is a subset $S \subseteq A$ with $|S| = \lceil 1/\rho \rceil$. Since $1/\lceil 1/\rho \rceil \leq \rho$, the uniform distribution on S is a $(0, \rho)$-improvising distribution. Since this distribution has finite support and rational probabilities, there is an expected finite-time probabilistic algorithm sampling from it, and this is a $(0, \rho)$-improviser. If instead $N < 1/\rho$, defining $M = \lceil 1/\rho \rceil - N$ we have $M \geq 1$. Since $|I| \geq \lceil 1/\rho \rceil = N + M$, there are disjoint subsets $S \subseteq A$ and $T \subseteq I$ with $|S| = N$ and $|T| = M$. Let D be the distribution on $S \cup T$ where each element of S has probability ρ and each element of T has probability $(1 - \rho N)/M = (1 - \rho N)/\lceil (1/\rho) - N \rceil = (1 - \rho N)/\lceil (1 - \rho N)/\rho \rceil \leq \rho$. Then $\Pr[w \in A | w \leftarrow D] \geq \rho N \geq 1 - \epsilon$, so D is a (ϵ, ρ)-improvising distribution. As above there is an expected finite-time probabilistic algorithm sampling from D, and this is an (ϵ, ρ)-improviser.
(c)⇒(a): Immediate. ◀

▶ Remark. In fact, whenever $\mathcal{C}$ is feasible, the construction in the proof of Theorem 3.1 gives an improviser which works in nearly the most trivial possible way: it has two finite lists S and T, flips a (biased) coin to decide which list to use, and then returns an element of that list uniformly at random.

A consequence of this characterization is that when there are infinitely-many admissible improvisations, there is an improviser with zero error probability:

▶ **Corollary 3.2.** *If A is infinite, $(\mathcal{I}, \alpha, 0, \rho)$ is feasible for any $\rho \in (0, 1] \cap \mathbb{Q}$.*

In addition to giving conditions for feasibility, Theorem 3.1 yields an algorithm which is guaranteed to find an improviser for any feasible CI problem.

▶ **Corollary 3.3.** *If $\mathcal{C}$ is feasible, an improviser for $\mathcal{C}$ may be found by an effective procedure.*

Proof. The sets I and A are clearly computably enumerable, since α is computable. We enumerate I and A until enough elements are found to perform the construction in Theorem 3.1. Since $\mathcal{C}$ is feasible, the theorem ensures this search will terminate. ◀

We cannot give an upper bound on the time needed by this algorithm without knowing something about the admissibility predicate α. Therefore although as noted in the remark above whenever there are improvisers at all there is one of a nearly-trivial form, actually finding such an improviser could be difficult. In fact, it could be faster to generate an improviser which is *not* of this form, as seen for example in Sec. 4.

▶ **Corollary 3.4.** *The set of feasible CI instances is computably enumerable but not computable.*

Proof. Enumerability follows immediately from the previous Corollary. If checking whether $\mathcal{C}$ is feasible were decidable, then so would be checking if $|A| \geq (1-\epsilon)/\rho$, but this is undecidable since α can be an arbitrary computable predicate. ◀

4 Finite-Memory Admissibility Predicates

In order to bound the time needed to find an improviser, we must constrain the admissibility predicate α. Perhaps the simplest type of admissibility predicate is one which can be computed by a DFA, i.e., one such that there is some DFA $\mathcal{D}$ which accepts a word $w \in \Sigma^*$ iff $\alpha(w) = 1$. This captures the notion of a *finite-memory* admissibility predicate, where admissibility of a word can be determined by scanning the word left-to-right, only being able to remember a finite number of already-seen symbols. An example of a finite-memory predicate α is one such that $\alpha(w) = 1$ iff each subword of w of a fixed constant length satisfies some condition. By the pumping lemma, such predicates have the property that continually repeating some section of a word can produce an infinite family of improvisations, which could be a disadvantage if looking for "creative", non-repetitive improvisations. However, in applications such as music we impose a maximum length on improvisations, so this is not an issue.

▶ **Example 4.1** (Factor Oracles)**.** Recall that one way of measuring the divergence of an improvisation w generated by the factor oracle F built from a word w_{ref} is by counting the number of non-direct transitions that w causes F to take. Since DFAs cannot count without bound, we can use a sliding window of some finite size k. Then our admissibility predicate α can be that at any point as F processes w, the number of the previous k transitions which were non-direct lies in some interval $[\ell, h]$ with $0 \leq \ell \leq h \leq k$. This predicate can be encoded as a DFA of size $O(|F| \cdot 2^k)$ (see the Appendix for details). The size of the automaton grows exponentially in the size of the window, but for small windows it can be reasonable.

When the admissibility predicate is finite-memory and the automaton $\mathcal{I}$ is a DFA, there is an efficient procedure to test if an improviser exists and synthesize one if so. The construction is similar to that of Theorem 3.1, but avoids explicit enumeration of all improvisations to be put in the range of the improviser. To avoid enumeration we use a classic method of uniformly sampling from the language of a DFA $\mathcal{D}$ (see for example [13, 9]). The next few lemmas summarize the results we need, proofs being given in the Appendix for completeness. The first step is to determine the size of the language.

▶ **Lemma 4.2.** *If $\mathcal{D}$ is a DFA, $|L(\mathcal{D})|$ can be computed in polynomial time.*

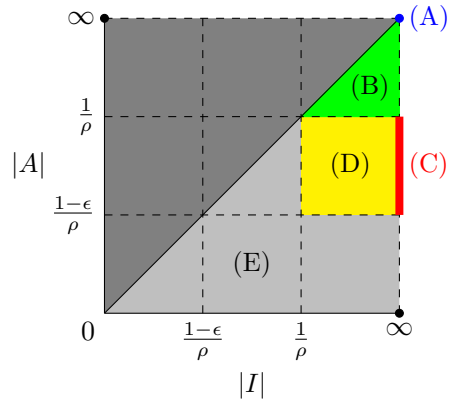


Figure 2 Cases for Theorem 4.5. The dark gray region cannot occur.

Once we know the size of $L(\mathcal{D})$ we can efficiently sample from it, handling infinite languages by sampling from a finite subset of a desired size.

▶ **Lemma 4.3.** *There is a polynomial $p(x, y)$ such that for any $N \in \mathbb{N}$ and DFA $\mathcal{D}$ with infinite language, there is a probabilistic algorithm S which uniformly samples from a subset of $L(\mathcal{D})$ of size N in expected time at most $p(|\mathcal{D}|, \log N)$, and which can be constructed in the same time.*

▶ **Lemma 4.4.** *There is a polynomial $q(x)$ such that for any DFA $\mathcal{D}$ with finite language, there is a probabilistic algorithm S which uniformly samples from $L(\mathcal{D})$ in expected time at most $q(|\mathcal{D}|)$, and which can be constructed in the same time.*

Using these sampling techniques, we have the following:

▶ **Theorem 4.5.** *The class of CI instances $\mathcal{C}$ where $\mathcal{I}$ is a DFA and α is computable by a DFA has a polynomial-time improvisation scheme.*

Proof. The proof considers five cases. We first define some notation. Let $\mathcal{D}$ denote the DFA giving α. Letting $\mathcal{A}$ be the product of $\mathcal{I}$ and $\mathcal{D}$, we have $A = L(\mathcal{A})$. This product can be computed in polynomial time since the automata are both DFAs, and $|\mathcal{A}|$ is polynomial in $|\mathcal{C}|$ and $|\mathcal{D}|$. In some of the cases below we will also use a DFA $\mathcal{B}$ which is the synchronous product of $\mathcal{I}$ and the complement of $\mathcal{A}$. Clearly $L(\mathcal{B}) = I \setminus A$, and the size of $\mathcal{B}$ and the time needed to construct it are also polynomial in $|\mathcal{C}|$ and $|\mathcal{D}|$.

Next we compute $|A| = |L(\mathcal{A})|$ and $|I| = |L(\mathcal{I})|$ in polynomial time using Lemma 4.2. There are now several cases (illustrated in Figure 2):

(A) $|A| = \infty$: Applying Lemma 4.3 to $\mathcal{A}$ with $N = \lceil 1/\rho \rceil$, we obtain a probabilistic algorithm S which uniformly samples from a subset of $L(\mathcal{A}) = A$ of size $\lceil 1/\rho \rceil$. Since $1/\lceil 1/\rho \rceil \leq \rho$, we have that S is a $(0, \rho)$-improviser and return it.

(B) $1/\rho \leq |A| < \infty$: Applying Lemma 4.4 to $\mathcal{A}$, we obtain a probabilistic algorithm S which uniformly samples from $L(\mathcal{A}) = A$. Since $1/|A| \leq \rho$, we have that S is a $(0, \rho)$-improviser and return it.

(C) $(1 - \epsilon)/\rho \leq |A| < 1/\rho$ and $|I| = \infty$: Applying Lemma 4.4 to $\mathcal{A}$ we obtain S as in the previous case. Defining $M = \lceil 1/\rho \rceil - |A|$, we have $\infty = |L(\mathcal{B})| > M \geq 1$. Applying Lemma 4.3 to $\mathcal{B}$ with $N = M$ yields a probabilistic algorithm S' which uniformly samples from a subset of $L(\mathcal{B}) = I \setminus A$ of size M. Let G be a probabilistic algorithm which with probability $\rho|A|$ executes S, and otherwise executes S'. Then since $L(\mathcal{A}) = A$ and

$L(\mathcal{B}) = I \setminus A$ are disjoint, every word generated by G has probability either $(\rho|A|)/|A| = \rho$ (if it is in A) or $(1-\rho|A|)/M = (1-\rho|A|)/\lceil(1-\rho|A|)/\rho\rceil \leq \rho$ (if it is in $I \setminus A$). Also, G outputs a member of A with probability $\rho|A| \geq 1-\epsilon$, so G is an (ϵ, ρ)-improviser and we return it.

(D) $(1-\epsilon)/\rho \leq |A| < 1/\rho \leq |I| < \infty$: As in the previous case, except obtaining S' by applying Lemma 4.4 to $\mathcal{B}$. Since $|I| \geq \lceil 1/\rho \rceil$, we have $|L(\mathcal{B})| = |I \setminus A| \geq M$ and so G as constructed above is an (ϵ, ρ)-improviser.

(E) $|I| < 1/\rho$ or $|A| < (1-\epsilon)/\rho$: By Theorem 3.1, $\mathcal{C}$ is not feasible, so we return $\perp$.

This procedure takes time polynomial in $|\mathcal{I}|$, $|\mathcal{D}|$, and $\log(1/\rho)$, so it is polynomial-time. Also, a fixed polynomial in these quantities bounds the expected runtime of the generated improviser, so the procedure is a polynomial-time improvisation scheme. ◀

Running Example. Recall that for our running example $\mathcal{C} = (\mathcal{I}, \alpha, 1/4, 1/4)$, we have $I = \{000, 001, 010, 100, 101\}$ and $A = \{000, 001, 101\}$. Since $|A| = 3$ and $|I| = 5$, we are in case (4) of Theorem 4.5. So our scheme uses Lemma 4.4 to obtain S and S' uniformly sampling from A and $I \setminus A = \{010, 100\}$ respectively. It returns a probabilistic algorithm G that executes S with probability $\rho|A| = 3/4$ and otherwise executes S'. So G returns 000, 001, and 101 with probability 1/4 each, and 010 and 100 with probability 1/8 each. The output distribution of G satisfies our conditions, so it is an improviser for $\mathcal{C}$.

5 More Complex Automata

While counting the language of a DFA is easy, in the case of an NFA it is much more difficult, and so there are unlikely to be polynomial-time improvisation schemes for more complex automata. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the classes of CI instances where $\mathcal{I}$ or α respectively are given by an NFA, and the other is given by a DFA. Then denoting by $\mathcal{N}$ either of these classes, we have (deferring full proofs from this section to the Appendix):

▶ **Theorem 5.1.** *Determining whether $\mathcal{C} \in \mathcal{N}$ is feasible is* #P*-hard.*

Proof sketch. The problem of counting the language of an NFA, which is #P-hard [14], is polynomially reducible to that of checking if $\mathcal{C} \in \mathcal{N}$ is feasible. ◀

▶ Remark. Determining feasibility of $\mathcal{N}$-instances is not a counting problem, so it is not #P-complete, but it is clearly in $\mathsf{P}^{\#\mathsf{P}}$: we construct the automata $\mathcal{I}$ and $\mathcal{A}$ as in Theorem 4.5 (now they can be NFAs), count their languages using #P, and apply Theorem 3.1.

▶ **Corollary 5.2.** *If there is a polynomial-time improvisation scheme for $\mathcal{N}$, then $\mathsf{P} = \mathsf{P}^{\#\mathsf{P}}$.*

This result indicates that in general, the control improvisation problem is probably intractable in the presence of NFAs. Some special cases could still be handled in practice: for example, if the NFA is very small it could be converted to a DFA. Another tractable case is where although one of $\mathcal{I}$ or $\mathcal{A}$ (as in Theorem 4.5) is an NFA, it has infinite language (this can clearly be detected in polynomial time). If $\mathcal{A}$ is an NFA with infinite language we can use case (1) of the proof of Theorem 4.5, since an NFA can be pumped in the same way as a DFA. If instead $\mathcal{A}$ is a DFA with finite language but $\mathcal{I}$ is an NFA with infinite language, one of the other cases (2), (3), or (5) applies, and in case (3) we can sample $I \setminus A$ by pumping $\mathcal{I}$ enough to ensure we get a string longer than any accepted by $\mathcal{A}$. Table 1 in Section 7 summarizes these cases.

For still more complex automata, the CI problem becomes even harder. In fact, it is impossible if we allow either $\mathcal{I}$ or α to be given by a PFA. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the classes of

CI instances where each of these respectively are given by a PFA, and the other is given by a DFA. Then letting $\mathcal{P}$ be either of these classes, we have:

▶ **Theorem 5.3.** *Determining whether $\mathcal{C} \in \mathcal{P}$ is feasible is undecidable.*

Proof sketch. Checking the feasibility of $\mathcal{C}$ amounts to counting the language of a PFA, but determining whether the language of a PFA is empty is undecidable [17, 8]. ◀

6 Symbolic Techniques

Previously we have assumed that the automata defining a control improvisation problem were given explicitly. However, in practice there may be insufficient memory to store full transition tables, in which case an implicit representation is required. This prevents us from using the polynomial-time improvisation scheme of Theorem 4.5, so we must look for alternate methods. These will depend on the type of implicit representation used. We focus on representations of DFAs and NFAs by propositional formulae, as used for example in bounded model checking [4].

▶ **Definition 6.1.** A *symbolic automaton* is a transition system over states $S \subseteq \{0,1\}^n$ and inputs $\Sigma \subseteq \{0,1\}^m$ represented by:

- a formula $\text{init}(\overline{x})$ which is true iff $\overline{x} \in \{0,1\}^n$ is an initial state,
- a formula $\text{acc}(\overline{x})$ which is true iff $\overline{x} \in \{0,1\}^n$ is an accepting state, and
- a formula $\delta(\overline{x}, \overline{a}, \overline{y})$ which is true iff there is a transition from $\overline{x} \in \{0,1\}^n$ to $\overline{y} \in \{0,1\}^n$ on input $\overline{a} \in \{0,1\}^m$.

A symbolic automaton accepts words in Σ^* according to the usual definition for NFAs.

Given a symbolic automaton, it is straightforward to generate a formula whose models correspond, for example, to accepting paths of at most a given length (see [4] for details). A SAT solver can then be used to find such a path. We refer to the length of the longest simple accepting path as the *diameter* of the automaton. This will be an important parameter in the runtime of our algorithms. In some cases an upper bound on the diameter is known ahead of time: for example, if we only want improvisations of up to some maximum length, and have encoded that constraint in $\mathcal{I}$. If the diameter is not known, it can be found iteratively with SAT queries asserting the existence of a simple accepting path of length n, increasing n until we find no such path exists. The diameter could be exponentially large compared to the symbolic representation, but this is a worst-case scenario.

Our approach for solving the control improvisation problem with symbolic automata will be to adapt the procedure of Theorem 4.5, replacing the counting and sampling techniques used there with ones that work on symbolic automata. For language size estimation we use the following:

▶ **Lemma 6.2.** *If $\mathcal{S}$ is a symbolic automaton with diameter D, for any $\tau, \delta > 0$ we can compute an estimate of $|L(\mathcal{S})|$ accurate to within a factor of $1+\tau$ in time polynomial in $|\mathcal{S}|$, D, $1/\tau$, and $\log(1/\delta)$ relative to an* NP *oracle.*

Sampling from infinite languages can be done by a direct adaptation of the method for explicit DFAs in Lemma 4.3.

▶ **Lemma 6.3.** *There is a polynomial $p(x,y,z)$ such that for any $N \in \mathbb{N}$ and symbolic automaton $\mathcal{Y}$ with infinite language and diameter D, there is a probabilistic oracle algorithm S^{NP} which uniformly samples from a subset of $L(\mathcal{Y})$ of size N in expected time at most $p(|\mathcal{Y}|, D, \log N)$ and which can be constructed in the same time.*

To sample from a finite language, we use techniques for almost-uniform generation of models of propositional formulae. In theory uniform sampling can be done exactly using a SAT solver [3], but the only algorithms which work in practice are *approximate* uniform generators such as UniGen [6]. This algorithm guarantees that the probability of returning any given model is within a factor of $1+\tau$ of the uniform probability, for any given $\tau > 6.84$ (the constant is for technical reasons specific to UniGen). UniGen can also do projection sampling, i.e., sampling where two models are considered identical if they agree on the set of variables being projected onto. Henceforth, for simplicity, we will assume we have a generic almost-uniform generator that can do projection, and will ignore the $\tau > 6.84$ restriction imposed by UniGen (although we might want to abide by this in practice in order to be able to use the fastest available algorithm). We assume that the generator runs in time polynomial in $1/\tau$ and the size of the given formula relative to an NP oracle, and succeeds with at least some fixed constant probability.

▶ **Lemma 6.4.** *There is a polynomial $q(x,y,z)$ such that for any $\tau > 0$ and symbolic automaton $\mathcal{Y}$ with finite language and diameter D, there is a probabilistic oracle algorithm S^{NP} which samples from $L(\mathcal{S})$ uniformly up to a factor of $1+\tau$ in expected time at most $q(|\mathcal{Y}|, D, 1/\tau)$, and which can be constructed in the same time.*

Now we can put these methods together to get a version of Theorem 4.5 for symbolic automata. The major differences are that this scheme requires an NP oracle, has some probability of failure (which can be specified), and returns an improviser with a slightly sub-optimal value of ρ. The proof generally follows that of Theorem 4.5, so we only sketch the differences here (see the Appendix for a full proof).

▶ **Theorem 6.5.** *There is a procedure that given any CI problem $\mathcal{C}$ where $\mathcal{I}$ and α are given by symbolic automata with diameter at most D, and any $\epsilon \in [0,1]$, $\rho \in (0,1]$, and $\tau, \delta > 0$, if $\mathcal{C}$ is $(\epsilon, \rho/(1+\tau))$-feasible returns an $(\epsilon, (1+\tau)^2(1+\epsilon)\rho)$-improviser with probability at least $1-\delta$. Furthermore, the procedure and the improvisers it generates run in expected time given by some fixed polynomial in $|\mathcal{C}|$, D, $1/\tau$, and $\log(1/\delta)$ relative to an* NP *oracle.*

Proof sketch. We first compute estimates E_A and E_I of $|A|$ and $|I|$ respectively using Lemma 6.2. Then we break into cases as in Theorem 4.5:

(A) $E_A = \infty$: As in case (1) of Theorem 4.5, using Lemma 6.3 in place of Lemma 4.3. We obtain a $(0,\rho)$-improviser.

(B) $1/\rho \le E_A < \infty$: As in case (2) of Theorem 4.5, using Lemma 6.4 in place of Lemma 4.4. Since we can do only approximate counting and sampling, we obtain a $(0,(1+\tau)^2\rho)$-improviser.

(C) $(1-\epsilon)/\rho \le E_A < 1/\rho$ and $E_I = \infty$: As in case (3) of Theorem 4.5, using Lemmas 6.3 and 6.4 in place of Lemmas 4.3 and 4.4. Our use of approximate counting/sampling means we obtain only an $(\epsilon,(1+\tau)^2\rho)$-improviser.

(D) $(1-\epsilon)/\rho \le E_A < 1/\rho \le E_I < \infty$: We cannot use the procedure in case (4) of Theorem 4.5, since it may generate an element of $L(\mathcal{B})$ with too high probability if our estimate E_A is sufficiently small. Instead we sample almost-uniformly from $L(\mathcal{A})$ with probability ϵ, and from $L(\mathcal{I})$ with probability $1-\epsilon$. This yields an $(\epsilon,(1+\tau)^2(1+\epsilon)\rho)$-improviser.

(E) $E_I < 1/\rho$ or $E_A < (1-\epsilon)/\rho$: We return $\perp$.

If $\mathcal{C}$ is $(\epsilon, \rho/(1+\tau))$-feasible, case (5) happens with probability less than δ by Theorem 3.1. Otherwise, we obtain an $(\epsilon,(1+\tau)^2(1+\epsilon)\rho)$-improviser. ◀

Therefore, it is possible to *approximately* solve the control improvisation problem when the automata are given by a succinct propositional formula representation. This allows working

Table 1 Complexity of the control improvisation problem when $\mathcal{I}$ and α are given by various different types of automata. The cell marked '-' is impossible since $L(\mathcal{A}) \subseteq L(\mathcal{I})$.

<table>
<tr><th colspan="2">α / $\mathcal{I}$</th><th>DFA</th><th colspan="2">NFA</th><th>PFA</th></tr>
<tr><th colspan="2"></th><th></th><th>$L(\mathcal{A}) = \infty$</th><th>$L(\mathcal{A}) < \infty$</th><th></th></tr>
<tr><td colspan="2">DFA</td><td colspan="2" rowspan="2">poly-time</td><td rowspan="2">#P-hard</td><td rowspan="3"></td></tr>
<tr><td rowspan="2">NFA</td><td>$L(\mathcal{I}) = \infty$</td></tr>
<tr><td>$L(\mathcal{I}) < \infty$</td><td>#P-hard</td><td>-</td><td></td></tr>
<tr><td colspan="2">PFA</td><td colspan="4">undecidable</td></tr>
</table>

with general NFAs, and very large automata that cannot be stored explicitly, but comes at the cost of using a SAT solver (perhaps not a heavy cost given the dramatic advances in the capacity of SAT solvers) and possibly having to increase ρ by a small factor.

7 Conclusion

In this paper, we introduced control improvisation, the problem of creating improvisers that randomly generate variants of words in the languages of automata. We gave precise conditions for when improvisers exist, and investigated the complexity of finding improvisers for several major classes of automata. In particular, we showed that the control improvisation problem for DFAs can be solved in polynomial time, while it is intractable in most cases for NFAs and undecidable for PFAs. These results are summarized in Table 1. Finally, we studied the case where the automata are presented symbolically instead of explicitly, and showed that the control improvisation problem can still be solved approximately using SAT solvers.

One interesting direction for future work would be to find other tractable cases of the control improvisation problem deriving from finer structural properties of the automata than just determinism. Extensions of the theory to other classes of formal languages, for instance context-free languages represented by pushdown automata or context-free grammars, are also worthy of study. Finally, we are investigating further applications, particularly in the areas of testing, security, and privacy.

Acknowledgements. The first three authors dedicate this paper to the memory of the fourth author, David Wessel, who passed away while it was being written. We would also like to thank Ben Caulfield, Orna Kupferman, Markus Rabe, and the anonymous reviewers for their helpful comments. This work is supported in part by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1106400, by the NSF Expeditions grant CCF-1139138, and by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

1 Gérard Assayag, Georges Bloch, Marc Chemillier, Benjamin Lévy, and Shlomo Dubnov. OMax. `http://repmus.ircam.fr/omax/home`, 2012.

2 Gérard Assayag and Shlomo Dubnov. Using factor oracles for machine improvisation. *Soft Comput.*, 8(9):604–610, 2004.

3 Mihir Bellare, Oded Goldreich, and Erez Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Information and Computation*, 163(2):510–526, 2000.

4 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, 1999.

5 Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

6 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Balancing scalability and uniformity in SAT witness generator. In *51st Design Automation Conference*, pages 1–6. ACM, 2014.

7 Loek Cleophas, Gerard Zwaan, and Bruce W. Watson. Constructing factor oracles. In *In Proceedings of the 3rd Prague Stringology Conference*, 2003.

8 Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs. In *30th Annual Symposium on Foundations of Computer Science*, pages 462–467. IEEE, 1989.

9 Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, and Sylvain Peyronnet. Uniform random sampling of traces in very large models. In *1st International Workshop on Random Testing*, pages 10–19. ACM, 2006.

10 Alexandre Donzé, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC)*, September 2014.

11 Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. Control improvisation. arXiv preprint arXiv:1411.0698, 2015.

12 Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *International Conference on Software Engineering (ICSE Future of Software Engineering)*. IEEE, May 2014.

13 Timothy Hickey and Jacques Cohen. Uniform random generation of strings in a context-free language. *SIAM Journal on Computing*, 12(4):645–655, 1983.

14 Sampath Kannan, Z. Sweedyk, and Steve Mahaney. Counting and random generation of strings in regular languages. In *Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 551–557. SIAM, 1995.

15 Stéphane Lafortune. Personal Communication, 2015.

16 Edward A. Lee. Personal Communication, 2013.

17 Masakazu Nasu and Namio Honda. Mappings induced by PGSM-mappings and some recursively unsolvable problems of finite probabilistic automata. *Information and Control*, 15(3):250–273, 1969.

18 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

19 R. Rowe. *Machine Musicianship*. MIT Press, 2001.

20 Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 2007.

A Provably Correct Sampler for Probabilistic Programs

Chung-Kil Hur[1], Aditya V. Nori[2], Sriram K. Rajamani[2], and Selva Samuel[3]

1 **Seoul National University, South Korea**
gil.hur@sf.snu.ac.kr
2 **Microsoft Research, US**
{adityan,sriram}@microsoft.com
3 **Carnegie Mellon University, US**
ssamuel@cs.cmu.edu

Abstract

We consider the problem of inferring the implicit distribution specified by a probabilistic program. A popular inference technique for probabilistic programs called Markov Chain Monte Carlo or MCMC sampling involves running the program repeatedly and generating sample values by perturbing values produced in "previous runs". This simulates a Markov chain whose stationary distribution is the distribution specified by the probabilistic program.

However, it is non-trivial to implement MCMC sampling for probabilistic programs since each variable could be updated at multiple program points. In such cases, it is unclear which values from the "previous run" should be used to generate samples for the "current run".

We present an algorithm to solve this problem for the general case and formally prove that the algorithm is correct. Our algorithm handles variables that are updated multiple times along the same path, updated along different paths in a conditional statement, or repeatedly updated inside loops, We have implemented our algorithm in a tool called Infer$^{\checkmark}$. We empirically demonstrate that Infer$^{\checkmark}$ produces the correct result for various benchmarks, whereas existing tools such as R2 and Stan produce incorrect results on several of these benchmarks.

1998 ACM Subject Classification D.2.4 [Software Engineering] Software/Program Verification

Keywords and phrases Probabilistic Programming, Program Correctness, Probabilistic Inference, Markov Chain Monte Carlo Sampling

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.475

1 Introduction

Recent years have seen a wide variety of languages for writing probabilistic programs, as well as tools and techniques for performing inference over these programs [7, 16, 24, 21, 8, 14, 22]. One of the main advantages of probabilistic programming is that it allows developers who are familiar with programming language notation, but unfamiliar with machine learning, to focus on the specification of the probabilistic model, and not worry about how to implement inference algorithms over the model. Probabilistic programming tools are able to automatically generate inference code from specifications written as probabilistic programs, thus reducing the degree of expertise required to implement a machine learning algorithm.

We focus on sampling-based inference, in particular, Metropolis-Hastings (MH) based sampling algorithms [3] for probabilistic programming languages. MH based sampling involves execution of the input program with the characteristic that the sample generated at the "current run" depends on the sample generated during the "previous run".

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 475–488

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In a probabilistic program, a variable can be assigned values more than once in a single path or assigned values along different paths, and it is unclear what "previous value" to use in generating a "current value" of the variable. For existing probabilistic programming tools such as R2 [22] and Stan [12], the notion of "previous value" used is incorrect (we give examples and more details later), and therefore these tools can produce incorrect results.

In this paper, we precisely define a sample drawn from the distribution represented by the program by specifying a big-step operational semantics for probabilistic programs. We show that this operational semantics is equivalent to the widely accepted denotational semantics of probabilistic programs [17].

Based on this notion of a sample, we then propose a simple MH algorithm for probabilistic programs where a variable can be updated at multiple program points. Our main insight is to track the ordered list of values that a variable gets assigned during a run, together with the distributions that were used to generate these values. We present a procedure to make use of values from such a list generated during a "previous run" to generate samples for the "current run", as well as calculate the quantities (such as acceptance ratio) that are needed for MH sampling. Complications arise because each run of the program can follow different paths with potentially different number of probabilistic assignments to a variable along each path resulting in lists of different lengths across different runs. Our algorithm handles all such cases.

We have implemented our algorithm in a tool called Infer$^{\checkmark}$, and compare it with existing probabilistic programming tools such as R2 and Stan. We prove formally that our algorithm correctly implements MH for probabilistic programs. We also demonstrate cases where existing tools produce incorrect results, whereas our algorithm produces correct results in all cases.

2 Overview

In this section, we first introduce probabilistic programs with an example, and then motivate our algorithm by describing the complications that arise while performing *correct* MH sampling-based inference for probabilistic programs.

Consider the probabilistic program in Figure 1 that is defined over two Boolean variables `x` and `y`. The program tosses two fair coins (modeled by calls to `Bernoulli(0.5)`) in lines 2 and 3, and assigns the outcomes to the variables `x` and `y` respectively. The observe statement `observe(x || y)` in line 4 blocks all executions of the program that do not satisfy the Boolean expression `(x || y)`. The meaning of this program is the distribution over its return expression (which is the tuple `(x, y)` conditioned by permitted executions of the program). This distribution is: $\Pr(\texttt{x} = \texttt{false}, \texttt{y} = \texttt{false}) = 0$, and $\Pr(\texttt{x} = \texttt{false}, \texttt{y} = \texttt{true}) = \Pr(\texttt{x} = \texttt{true}, \texttt{y} = \texttt{false}) = \Pr(\texttt{x} = \texttt{true}, \texttt{y} = \texttt{true}) = 1/3$.

Inference. Probabilistic inference is the task of determining the distribution implicitly specified by a probabilistic program. Inference can be performed by executing the program several times and averaging over the resulting samples. To do this efficiently, many probabilistic programming tools [8, 12, 22] employ Markov Chain Monte Carlo (MCMC) sampling algorithms [19] and their variants.

```
1: bool x, y;
2: x ~ Bernoulli(0.5);
3: y ~ Bernoulli(0.5);
4: observe(x || y);
5: return(x, y);
```

Figure 1 A simple probabilistic program.

To make this paper self-contained, we give a brief overview of the most basic form of an MCMC algorithm which is the Metropolis-Hastings

(MH) algorithm [19]. The MH algorithm takes a *target probability distribution* $T(\bar{x})$ as input, and returns samples that are distributed according to this distribution by performing the following steps:

1. First, a *proposal distribution* $Q(v_{old} \to v_{new})$ is used to pick a new value v_{new} for the variable $\bar{x}$ by appropriately perturbing its old value v_{old}.
2. Next, a parameter β called the *acceptance ratio* is computed. It is used to decide whether to accept or reject the new sampled value v_{new} for $\bar{x}$, and is defined as follows:

$$\beta = \min\left\{1, \frac{T(v_{new}) \times Q(v_{new} \to v_{old})}{T(v_{old}) \times Q(v_{old} \to v_{new})}\right\}$$

3. The sample is accepted if a random draw from a Bernoulli distribution with mean β (1 occurs with probability β and 0 occurs with probability $1 - \beta$) results in a 1, otherwise it is rejected.

The MH algorithm executes the above steps iteratively to generate samples.

A probabilistic program P denotes a target distribution T implicitly (see Section 3 for a formal definition of the target distribution denoted by a probabilistic program).

In order to perform MH sampling on a probabilistic program, we need to run the program P which results in the state being constructed incrementally, as P executes along a path. Along the path, the program encounters several probabilistic assignments to variables, and to generate a new value for each variable, we require the corresponding old value of the variable from the previous run of the program. Such an association between old and new values is easy if each variable is assigned only once, or if the program is a single path without branches or loops.

However, associating old values with new values is non-trivial for the general case. If the program has branches or loops, the previous value corresponding to a probabilistic assignment may come from an assignment in a different branch than the one currently being executed. Also, different branches may generate samples for the same variable from different distributions. Alternatively, a variable may have been assigned multiple times during execution of the previous run, and it is sometimes unclear which of these values is to be used to generate values for the current run. So, care must be taken to compute the MH acceptance ratio correctly. Due to these reasons, implementing the above 3 steps of the MH algorithm for a probabilistic program is non-trivial.

We illustrate the difficulties with implementing a correct sampling algorithm via the following examples.

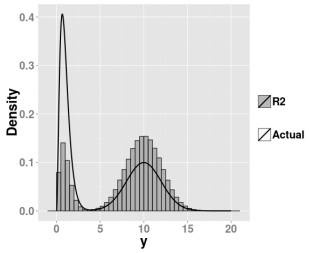
Example 1 (mixtures). Consider the probabilistic program shown in Figure 2(a). The program is defined over two variables `x` and `y`. The variable `y` is drawn from a mixture of a Gaussian distribution and a Gamma distribution (specified by the if-then-else statement in lines 3–6), whereas the variable `x` is drawn from another Gaussian distribution (line 2), and determines the mixture proportion.

Suppose we perform MH sampling for this example. Assume that during run n, the program follows the path 1, 2, 3, 4, 7. That is, `x` was assigned a value greater than 0, say 0.1, and the "then" branch of the conditional statement was taken, and `y` was assigned a value say 9.6 from the distribution `Gaussian(10, 2)`.

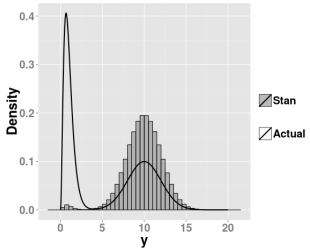
Next, we consider how to execute run $n + 1$ using MH sampling. During run $n + 1$, at line 2, in order to generate a current value for `x`, we need to propose a value for `x` using a proposal distribution Q_x centered around the old value of `x`, namely 0.1, and calculate the

```
1: double x, y;
2: x ~ Gaussian(0, 1);
3: if (x > 0) then
4:   y ~ Gaussian(10, 2);
5: else
6:   y ~ Gamma(3, 3);
7: return y;
```

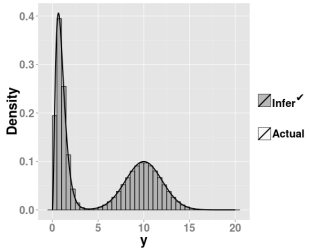
(a)



(b)



(c)



(d)

Figure 2 A probabilistic program for a mixture model together with inference results.

parameter β_x as the ratio described earlier. Now suppose the current value of `x` so chosen is -0.05 which is less than 0, then the "else" branch is taken, and we need to produce a current value for `y`.

How should we now generate the current value for `y`? Prior work such as [29] use the value of y from the most recent run which took the "else" branch, say run $n-3$ (assuming runs $n-2$ and $n-1$ took the "then" branch) and use that value to generate the current value of `y`. The probabilistic programming tools R2 [22] and Stan [12] follow the same algorithm.

However, if a value from a run other than the previously accepted run is used, then this would result in the algorithm converging to the incorrect distribution. This is shown in the plots in Figure 2. Plot 2(b) depicts the distribution inferred by R2 for the program in Figure 2(a), and plot 2(c) shows the distribution computed by Stan for the same program. The density function of the correct distribution for this example is shown by the line graph labelled `Actual` in each of the plots in Figure 2.

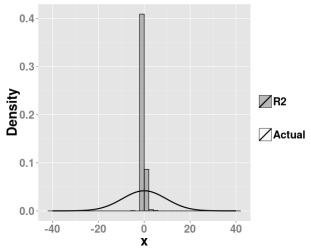
Example 2 (loop). Consider the probabilistic program for a hierarchical model shown in Figure 3(a). The loop in lines 4–7 constructs a series of Gaussian distributions with the final answer also being a Gaussian distribution (shown by the line graphs labelled `Actual` in the plots of Figure 3). The variable `x` is assigned 11 times during an execution of the program.

Tools like R2 and Stan use the last value that `x` was assigned in the previous run to generate each of the 11 values in the current run. So, these tools produce incorrect results as shown by the density histogram plots 3(b) and 3(c), respectively. To perform MH sampling correctly, it is necessary to record each of the 11 values generated for `x` and use the corresponding previous value to propose a new value in the current run.

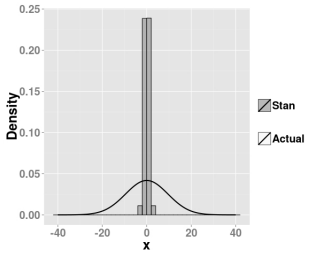
In addition to keeping track of previous values correctly, it is also important to record the distributions in the corresponding sampling statements whose execution generated those values. This is necessary so that the density of the previous value may be computed w.r.t. the correct target distribution while computing the acceptance ratio.

```
1: double x;
2: int i = 0;
3: x ~ Gaussian(0, 1);
4: while (i < 10) do {
5:   x ~ Gaussian(x, 3);
6:   i = i+1;
7: }
8: return x;
```

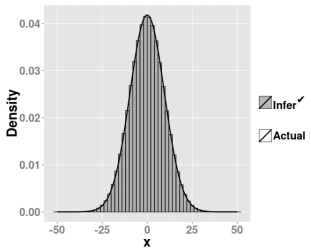
(a)



(b)



(c)



(d)

Figure 3 A probabilistic program for a hierarchical model together with inference results.

Algorithm. Motivated by the above examples, we desire to come up with an algorithm which correctly chooses the appropriate value from the previous run to be used to propose values for the current run, and computes the correct acceptance ratio in all possible scenarios.

Our sampling algorithm, called INFER$^{\checkmark}$, is based on a sampling-based operational semantics for probabilistic programs that specifies the meaning of a sample by clearly setting out all the values generated in an execution that would need to be tracked. The operational semantics also specifies how to compute the probability density for the sample generated in an execution so that the MH acceptance ratio may be computed correctly. In addition, INFER$^{\checkmark}$ uses samples from only the previously accepted run to propose new values in the current run. We also prove that by doing this, INFER$^{\checkmark}$ computes the correct distribution specified by any input probabilistic program. We informally describe the main ideas behind the algorithm next. A more complete and formal description is given in Sections 3 and 4.

INFER$^{\checkmark}$ uses a list for each variable in the program to keep track of the probabilistic assignments to that variable during a single run of the program (as opposed to the standard variable to value mapping). Each element of the list is a pair whose first element is the value generated for the corresponding variable when a sampling statement was encountered during the execution of the program. The second element of the pair is used to store information about the distribution used in the sampling statement and its parameters. This information about distributions is stored in order to compute the MH acceptance ratio correctly.

Specifically, INFER$^{\checkmark}$ maintains two lists for each sampled variable x in the program:

1. The first list l_{PRE} is used to store the samples generated for x together with the distribution information at the corresponding sampling statement executed in the most recent run which produced a sample that was accepted. The values in this list are used to propose new values for x. The distribution information is used to update the MH acceptance ratio.
2. The second list l_{CUR} is used to store the samples produced for x paired with the distribution information at the corresponding sampling statement executed in the current run of the

program. When a sample is accepted, l_{PRE} is assigned the value of l_{CUR}. Otherwise, l_{CUR} is discarded, and constructed again as the program is executed and new values are proposed for x.

We now informally show how INFER$^{\checkmark}$ correctly works for the previous examples.

Example 1 (mixtures). For the program in Figure 2(a), the variables `x` and `y` are sampled only once in every run of the program. Therefore, the l_{PRE} and l_{CUR} lists for both `x` and `y` will each contain one element at the end of every run. The sample value in the l_{PRE} and l_{CUR} lists for `x` is produced by the execution of the probabilistic assignment statement in line 2. The distribution information for `x` would note that the distribution at line 2 is a standard Gaussian distribution.

On the other hand, the sample value and distribution information in the l_{PRE} and l_{CUR} lists for `y` can be generated either by executing line 4 (if the "then" branch is taken) or line 6 (if the "else" branch is taken). This ensures that correct previous values are used to propose new values of `y` even when different branches are taken in different runs of the program.

Storing the distribution information also enables the computation of the density of the previous sample w.r.t. to the correct target distribution which is needed for calculating the MH acceptance ratio. By doing this, INFER$^{\checkmark}$ is able to produce the correct answer as shown by the plot in Figure 2(d).

Example 2 (loop). The program in Figure 3(a) contains one random variable `x` which is sampled 11 times during an execution (one at line 3 and ten at line 5). Thus, the l_{PRE} and l_{CUR} lists for `x` will each contain 11 elements at the end of every run.

The first value in l_{PRE} is generated by the probabilistic assignment statement in line 3 in the previous run. This value is used to propose a new value for `x` when line 3 is executed in the current run. This new proposed value is added as the first element of the first pair in the list l_{CUR} for `x`. The corresponding distribution information in both the lists specifies that the target is the standard Gaussian distribution.

Similarly, the other 10 values in l_{PRE} come from the execution of the probabilistic assignment statement in line 5 during the iterations of the while loop in the previous run, and are used to propose a new value for `x` when line 5 is executed in the corresponding iteration in the current run. Each proposed value for `x` is added to the list l_{CUR} as the first element of the corresponding pair. The distribution information for each of these pairs in both lists specifies that the target distribution is a Gaussian distribution whose mean is the value of x before line 5 is executed in that iteration and whose standard deviation is 3.

As noted earlier, the distribution information is used to compute the density of the previous samples w.r.t. to the correct target distribution to enable the correct computation of the MH acceptance ratio. INFER$^{\checkmark}$ computes the correct distribution for this example also as can be seen in the plot 3(d).

Notice that we only maintain the sequence of values generated for a variable during an execution. It is not necessary to keep track of the program points which produced these values. Also, note that different runs of the program may produce lists of different lengths for a particular variable, since different runs can follow different paths in the program.

If the list l_{PRE} for a given variable `x` has *fewer* elements than those needed to produce values for the current run, then, for the extra samples that are produced in the current run, the algorithm resorts to *Metropolis independent sampling* [10, 27, 18]. It is a modification to the MH algorithm in which the proposal distribution is independent of the previous sample value. The MH acceptance ratio is also updated appropriately.

$x \in \text{Vars}$
$\mathbf{uop} ::= \cdots$ C unary operators
$\mathbf{bop} ::= \cdots$ C binary operators
$\varphi, \psi ::= \cdots$ logical formula

$\mathcal{E} ::=$ expressions
$\mid x$ variable
$\mid c$ constant
$\mid \mathcal{E}_1 \ \mathbf{bop} \ \mathcal{E}_2$ binary operation
$\mid \mathbf{uop} \ \mathcal{E}$ unary operation

$\mathcal{S} ::=$ statements
$\mid \texttt{skip}$ skip
$\mid x = \mathcal{E}$ deterministic assignment
$\mid x \sim \texttt{Dist}(\bar{\theta})$ probabilistic assignment
$\mid \texttt{observe}(\varphi)$ observe
$\mid \mathcal{S}_1; \mathcal{S}_2$ sequential composition
$\mid \texttt{if } \mathcal{E} \texttt{ then } \mathcal{S}_1 \texttt{ else } \mathcal{S}_2$ conditional composition
$\mid \texttt{while } \mathcal{E} \texttt{ do } \mathcal{S}$ while−do loop

$\mathcal{P} ::= \mathcal{S} \texttt{ return } (\mathcal{E}_1, \ldots, \mathcal{E}_n)$ program

Figure 4 Syntax of PROB.

- Unnormalized Semantics for Statements
  $[\![\mathcal{S}]\!] \in (\Sigma \to [0,1]) \to \Sigma \to [0,1]$

$$
\begin{aligned}
[\![\texttt{skip}]\!](f)(\sigma) &:= f(\sigma) \\
[\![x = \mathcal{E}]\!](f)(\sigma) &:= f(\sigma[x \leftarrow \sigma(\mathcal{E})]) \\
[\![x \sim \texttt{Dist}(\bar{\theta})]\!](f)(\sigma) &:= \int_{v \in \text{Val}} \texttt{Dist}(\sigma(\bar{\theta}))(v) \times f(\sigma[x \leftarrow v])\, dv \\
[\![\texttt{observe}(\varphi)]\!](f)(\sigma) &:= \begin{cases} f(\sigma) & \text{if } \sigma(\varphi) = \texttt{true} \\ 0 & \text{otherwise} \end{cases} \\
[\![\mathcal{S}_1; \mathcal{S}_2]\!](f)(\sigma) &:= [\![\mathcal{S}_1]\!]([\![\mathcal{S}_2]\!](f))(\sigma) \\
[\![\texttt{if } \mathcal{E} \texttt{ then } \mathcal{S}_1 \texttt{ else } \mathcal{S}_2]\!](f)(\sigma) &:= \begin{cases} [\![\mathcal{S}_1]\!](f)(\sigma) & \text{if } \sigma(\mathcal{E}) = \texttt{true} \\ [\![\mathcal{S}_2]\!](f)(\sigma) & \text{otherwise} \end{cases} \\
[\![\texttt{while } \mathcal{E} \texttt{ do } \mathcal{S}]\!](f)(\sigma) &:= \sup_{n \geq 0} [\![\texttt{while } \mathcal{E} \texttt{ do}_n \mathcal{S}]\!](f)(\sigma)
\end{aligned}
$$

  where
  $\texttt{while } \mathcal{E} \texttt{ do}_0 \mathcal{S} = \texttt{observe}(\texttt{false})$
  $\texttt{while } \mathcal{E} \texttt{ do}_{n+1} \mathcal{S} = \texttt{if } \mathcal{E} \texttt{ then } (\mathcal{S}; \texttt{while } \mathcal{E} \texttt{ do}_n \mathcal{S}) \texttt{ else } (\texttt{skip})$

- Normalized Semantics for Programs
  $[\![\mathcal{S} \texttt{ return } (\mathcal{E}_1, \ldots, \mathcal{E}_n)]\!] \in (\mathbb{R}^n \to [0,1]) \to [0,1]$

$$
[\![\mathcal{S} \texttt{ return } (\mathcal{E}_1, \ldots, \mathcal{E}_n)]\!](f) := \frac{[\![\mathcal{S}]\!](\lambda\sigma.\ f(\sigma(\mathcal{E}_1), \ldots, \sigma(\mathcal{E}_n)))(\bot)}{[\![\mathcal{S}]\!](\lambda\sigma.\ 1)(\bot)}
$$

  where $\bot$ denotes the default initial state.

Figure 5 Denotational Semantics of PROB.

On the other hand, if the list l_{PRE} for a given variable `x` has *more* elements than those needed to produce values for the current run, then, the extra values in l_{PRE} are used to produce some adjustments in the MH acceptance ratio β. These details are explained in Section 4.

3 Probabilistic Programs

We consider a probabilistic programming language called PROB [13] whose syntax is formally described in Figure 4. A PROB program consists of statements and a return expression. Variables have base types such as bool, int, float and double, and expressions include variables, constants, binary and unary operations. Statements include primitive statements (skip, deterministic assignment, probabilistic assignment, observe) and composite statements (sequential composition, conditionals and loops). Features such as arrays, pointers, structures and function calls can be incorporated in the language, but for the sake of brevity, we omit these features from the core language.

A popular choice of the specification of formal semantics of probabilistic programs is the denotational semantics introduced by Kozen in [17]. This is summarized in Figure 5. The denotational semantics specifies the meaning of a probabilistic program by defining the joint distribution over the output state of the program, where a state σ of a program is a partial

$$\frac{}{(\texttt{skip}, \sigma) \Downarrow^{\epsilon} (1, \sigma)} \qquad \frac{}{(x = \mathcal{E}, \sigma) \Downarrow^{\epsilon} (1, \sigma[x \leftarrow \sigma(\mathcal{E})])} \qquad \frac{\sigma(\varphi) = \texttt{true}}{(\texttt{observe}(\varphi), \sigma) \Downarrow^{\epsilon} (1, \sigma)}$$

$$\frac{v \in \text{Val} \qquad p = \texttt{Dist}(\sigma(\bar{\theta}))(v) > 0}{(x \sim \texttt{Dist}(\bar{\theta}), \sigma) \Downarrow^{x \mapsto [v]} (p, \sigma[x \leftarrow v])} \qquad \frac{(\mathcal{S}_1, \sigma) \Downarrow^{s_1} (p_1, \sigma_1) \qquad (\mathcal{S}_2, \sigma_1) \Downarrow^{s_2} (p_2, \sigma_2)}{(\mathcal{S}_1; \mathcal{S}_2, \sigma) \Downarrow^{s_1 \cdot s_2} (p_1 \times p_2, \sigma_2)}$$

$$\frac{\sigma(\mathcal{E}) = \texttt{true} \qquad (\mathcal{S}_1, \sigma) \Downarrow^{s_1} (p_1, \sigma_1)}{(\texttt{if}\ \mathcal{E}\ \texttt{then}\ \mathcal{S}_1\ \texttt{else}\ \mathcal{S}_2, \sigma) \Downarrow^{s_1} (p_1, \sigma_1)} \qquad \frac{\sigma(\mathcal{E}) = \texttt{false} \qquad (\mathcal{S}_2, \sigma) \Downarrow^{s_2} (p_2, \sigma_2)}{(\texttt{if}\ \mathcal{E}\ \texttt{then}\ \mathcal{S}_1\ \texttt{else}\ \mathcal{S}_2, \sigma) \Downarrow^{s_2} (p_2, \sigma_2)}$$

$$\frac{\sigma(\mathcal{E}) = \texttt{false}}{(\texttt{while}\ \mathcal{E}\ \texttt{do}\ \mathcal{S}, \sigma) \Downarrow^{\epsilon} (1, \sigma)} \qquad \frac{\sigma(\mathcal{E}) = \texttt{true} \qquad (\mathcal{S}; \texttt{while}\ \mathcal{E}\ \texttt{do}\ \mathcal{S}, \sigma) \Downarrow^{s} (p, \sigma)}{(\texttt{while}\ \mathcal{E}\ \texttt{do}\ \mathcal{S}, \sigma) \Downarrow^{s} (p, \sigma)}$$

where

$$\begin{aligned} \epsilon &= \lambda u.\, [\,] \\ x \mapsto [v] &= \lambda u.\, \text{if } u = x \text{ then } [v] \text{ else } [\,] \\ s_1 \cdot s_2 &= \lambda u.\, s_1(u) +\!\!+ s_2(u), \text{ and } +\!\!+ \text{ denotes list concatenation} \end{aligned}$$

Figure 6 Sampling-based operational semantics of Prob.

valuation of all its variables. The set of all states (possibly infinite) is denoted by Σ.

However, in order to design an MCMC algorithm, we need to have a distribution over program executions. To this end, we introduce a big-step operational semantics for Prob. The operational semantics defines the probability density of a sample. It is interesting to note here that each sample (a sequence of program states) uniquely determines a program execution. We will also assume that all program executions terminate with probability 1.

The operational semantics thus gives rise to a distribution over these program executions. From this distribution, the distribution over the output state can be derived by marginalizing out the intermediate values. We also show that this distribution is the same as that defined by the denotational semantics.

Sampling-based Operational Semantics. We now inductively define the sampling-based operational semantics of Prob in Figure 6, which will form the basis of our MH-based sampling algorithm presented in Section 4.

The relation $(\mathcal{S}, \sigma) \Downarrow^{s} (p, \sigma')$ intuitively means that if we run the program statement $\mathcal{S}$ with initial state σ, it may internally draw a sample $s \in \mathbb{S}$ from the associated distributions and terminate with output state σ', with probability density p. Here the sample space $\mathbb{S}$ is defined as $\Gamma \to \text{List}(\text{Val})$ with Γ the set of variables and $\text{Val} = \mathbb{Z} \uplus \mathbb{R}$. Notice that, as explained earlier, a sample consists of a *list* of values associated with each random variable.

All rules of the sampling-based operational semantics are standard except for probabilistic assignment, observe, and sequential composition statements. The probabilistic assignment statement draws a sample v from the given distribution with the associated density p and sets the variable x to v. The observe statement proceeds only when the given condition φ is met. The sequential composition statement executes the sub-statements in order, and then multiplies the associated densities and concatenates the generated samples.

4 Algorithm

Given a program $\mathcal{P}$ written in Prob, and κ (the number of samples to be generated) as input, Infer$^{\checkmark}$ (shown in Algorithm 1) returns a sequence of samples from the distribution specified by $\mathcal{P}$. Note that the parameter κ controls the accuracy of the algorithm—the greater the number of samples generated by the algorithm, the better is the approximation to the actual distribution specified by $\mathcal{P}$. Infer$^{\checkmark}$ uses standard ideas from MH sampling, which were reviewed in Section 2. The program $\mathcal{P}$ is executed for κ times, and every execution produces

Algorithm 1 $\textsc{Infer}^{\checkmark}(\mathcal{P}, \kappa)$

Input: A Prob program $\mathcal{P}$, and κ, the number of samples to be generated.
Output: Samples from the distribution specified by $\mathcal{P}$.

```
1: Ω := [], Θ_ACC := ϵ
2: for i = 1 to κ do
3:    β := 1.0, Θ_PRE := Θ_ACC, Θ := ϵ
4:    σ := ⊥, ret := ()
5:    S return(E_1, ..., E_n) := P
6:    (σ, β, Θ_PRE, Θ) := EVAL(S, σ, β, Θ_PRE, Θ)
7:    ret := (σ(E_1), ..., σ(E_n))
8:    for x in Γ do                          (* Γ is the set of variables in P *)
9:       while Θ_PRE(x) ≠ [] do
10:         (v_PRE, Δ_PRE) := HEAD(Θ_PRE(x))
11:         Θ_PRE(x) := TAIL(Θ_PRE(x))
12:         β := β × PROP(Δ_PRE)(v_PRE) / Δ_PRE(v_PRE)
13:      end while
14:   end for
15:   if ret ≠ () ∧ (Ω = [] ∨ β ≥ 1 ∨ BERNOULLI(β)) then
16:      Ω := ret :: Ω
17:      Θ_ACC := Θ
18:   else
19:      if Ω ≠ [] then Ω := HEAD(Ω) :: Ω endif
20:   end if
21: end for
22: return Ω
```

a sample (as defined by the operational semantics), and a value β which determines whether the sample will be accepted or rejected.

In line 1 of Algorithm 1, $\textsc{Infer}^{\checkmark}$ initializes variables Ω and Θ_{ACC}. The list Ω is initialized to an empty list, and is used to store the values of the return expression at the end of each accepted run of the program. The map Θ_{ACC} is used to store the most recently accepted sample and is initialized to the empty map. Θ_{ACC} maps each sampled variable to a list of pairs generated for that variable in the last accepted execution of the program. The first element of each pair in the list is the sample value v that is assigned to the variable. The second element is the probability distribution from which the value v of the variable is drawn and is used to compute the MH acceptance ratio β. Thus, the samples generated by Algorithm 1 conform to the definition of the sample specified by the operational semantics.

Lines 2–21 generate κ samples—each sample is either accepted or rejected based on the value of the parameter β (lines 15–20 which encode the standard MH acceptance criterion). Note that the first sample (represented by the test for empty Ω in line 15) is always accepted. If a sample is rejected, then the program sample generated from the previous execution of $\mathcal{P}$ is added to Ω (line 19). The maps Θ_{PRE} and Θ (initialized in line 3) have the same type as Θ_{ACC}. The map Θ_{PRE} is initialized to the previously accepted sample and is used to determine the proposal distribution to use at each sampling statement and to compute the acceptance ratio β correctly. The map Θ is used to build up the sample for the current execution of the program.

Note that $\textsc{Infer}^{\checkmark}$ has the same high level structure as the standard MH procedure. The recursive procedure Eval (called in line 6) operates over the syntactic structure of the program $\mathcal{P}$. It defines the *transitions* of the Markov chain constructed by Algorithm 1.

The procedure Eval is described by the rules in Figure 7. Given a statement $\mathcal{S}$, a program state σ (which is a partial map from variables to values), a parameter β (which is used to decide whether the sample generated is to be accepted or rejected), the map Θ_{PRE} and the map Θ, the procedure $\textsc{Eval}(\mathcal{S}, \sigma, \beta, \Theta_{\text{PRE}}, \Theta)$ computes new values for σ, β, Θ_{PRE} and Θ obtained after executing statement $\mathcal{S}$.

$$
\begin{array}{lcl}
\textsc{Eval}(x = \mathcal{E}, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & = & (\sigma[x \leftarrow \sigma(\mathcal{E})], \beta, \Theta_{\text{PRE}}, \Theta) \\
\textsc{Eval}(\texttt{skip}, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & = & (\sigma, \beta, \Theta_{\text{PRE}}, \Theta) \\
\textsc{Eval}(\mathcal{S}_1; \mathcal{S}_2, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & = & \textbf{let } (\sigma', \beta', \Theta'_{\text{PRE}}, \Theta') = \textsc{Eval}(\mathcal{S}_1, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) \textbf{ and} \\
 & & \textbf{let } (\sigma'', \beta'', \Theta''_{\text{PRE}}, \Theta'') = \textsc{Eval}(\mathcal{S}_2, \sigma', \beta', \Theta'_{\text{PRE}}, \Theta') \textbf{ in} \\
 & & (\sigma'', \beta'', \Theta''_{\text{PRE}}, \Theta'') \\
\textsc{Eval}(\texttt{if } \mathcal{E} \texttt{ then } \mathcal{S}_1 \texttt{ else } \mathcal{S}_2, & = & \textbf{if } \sigma(\mathcal{E}) \textbf{ then } \textsc{Eval}(\mathcal{S}_1, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) \\
\quad \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & & \textbf{else } \textsc{Eval}(\mathcal{S}_2, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) \\
\textsc{Eval}(x \sim \texttt{Dist}(\bar{\theta}), \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & = & \textbf{let } \Delta = \texttt{Dist}(\sigma(\bar{\theta})) \textbf{ and} \\
 & & \textbf{let } (v, \beta', \Theta'_{\text{PRE}}) = \\
 & & \textbf{if } \Theta_{\text{PRE}}(x) = [\,] \textbf{ then} \\
 & & \quad \textbf{let } v \sim \textsc{Prop}(\Delta) \textbf{ and} \\
 & & \quad \textbf{let } \beta' = \beta \times \frac{\Delta(v)}{\textsc{Prop}(\Delta)(v)} \textbf{ in } (v, \beta', \Theta_{\text{PRE}}) \\
 & & \textbf{else} \\
 & & \quad \textbf{let } (v_{\text{PRE}}, \Delta_{\text{PRE}}) = \textsc{Head}(\Theta_{\text{PRE}}(x)) \textbf{ and} \\
 & & \quad \textbf{let } v \sim \textsc{Prop}(\Delta, v_{\text{PRE}}) \textbf{ and} \\
 & & \quad \textbf{let } \beta' = \beta \times \frac{\Delta(v) \times \textsc{Prop}(\Delta_{\text{PRE}}, v)(v_{\text{PRE}})}{\Delta_{\text{PRE}}(v_{\text{PRE}}) \times \textsc{Prop}(\Delta, v_{\text{PRE}})(v)} \textbf{ in} \\
 & & \quad (v, \beta', \textsc{Tail}(\Theta_{\text{PRE}})) \\
 & & \textbf{in} \\
 & & (\sigma[x \leftarrow v], \beta', \Theta'_{\text{PRE}}, \Theta(x) +\!\!+ [(v, \Delta)]) \\
\textsc{Eval}(\texttt{observe } (\varphi), \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & = & \textbf{if } \sigma(\varphi) \textbf{ then } (\sigma, \beta, \Theta_{\text{PRE}}, \Theta) \textbf{ else } (\sigma, 0, \Theta_{\text{PRE}}, \Theta) \\
\textsc{Eval}(\texttt{while } \mathcal{E} \texttt{ do } \mathcal{S}, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) & = & \textbf{let } (\sigma', \beta', \Theta'_{\text{PRE}}, \Theta') = \textsc{Eval}(\mathcal{S}, \sigma, \beta, \Theta_{\text{PRE}}, \Theta) \textbf{ in} \\
 & & \textbf{if } \sigma(\neg\mathcal{E}) \textbf{ then} \\
 & & \quad (\sigma, \beta, \Theta_{\text{PRE}}, \Theta) \\
 & & \textbf{else} \\
 & & \quad \textsc{Eval}(\textbf{while } \mathcal{E} \texttt{ do } \mathcal{S}, \sigma', \beta', \Theta'_{\text{PRE}}, \Theta')
\end{array}
$$

Figure 7 Given a statement $\mathcal{S}$ and parameters σ, β, Θ_{PRE}, and Θ, EVAL computes a tuple of parameters evaluated over $\mathcal{S}$ which are used by Algorithm 1 .

For instance, consider the assignment statement $x = \mathcal{E}$. Upon executing this from a state σ, we obtain the state $\sigma[x \leftarrow \sigma(\mathcal{E})]$, i.e., the value corresponding to the variable x in σ is set to the evaluation of $\mathcal{E}$ over σ. The values of β, Θ_{PRE} and Θ remain unchanged. The rules for the other statements also proceed in a similar manner.

The rule for the probabilistic assignment statement "$x \sim \texttt{Dist}(\bar{\theta})$" in EVAL specifies how to generate a sample and update β appropriately. As seen in Figure 7, in this case, first the variable Δ is set to the probability distribution function $\texttt{Dist}(\bar{\theta})$ with respect to the current state σ (*i.e.,* the parameters of the distribution $\bar{\theta}$ are evaluated at the state σ). Next, a value v is sampled, and an update to β is performed based on the following conditions:

Previous value for x is not available. This condition is represented by the predicate $\Theta_{\text{PRE}}(x) = [\,]$, which says that there are no values associated with the variable x from the previous execution of $\mathcal{P}$. A new value v for x is drawn from a proposal distribution $\textsc{Prop}(\Delta)$ that only depends on the target distribution Δ. In essence, this is the *Metropolized independent sampling algorithm* [18] which is a modification to the MH algorithm in which the proposal distribution is independent of the previous sample value. There are a number of choices for the proposal distribution, and in our implementation we set $\textsc{Prop}(\Delta)$ to be the target distribution Δ.

The updated value β' of β is the usual update for an MH algorithm. Intuitively, it is helpful to think of the previous sample value as a special value that is drawn from a distribution which produces this value with probability 1. If the proposal for this distribution is taken to be the distribution itself, then the MH acceptance ratio for x reduces to $\frac{\Delta(v)}{\textsc{Prop}(\Delta)(v)}$.

Previous value for x is available. This means that the list associated with the variable x in the map Θ_{PRE} is not empty. In particular, the previous value v_{PRE} for x and the probability density function Δ_{PRE} from which v_{PRE} is drawn are at the head of the list for x in the map Θ_{PRE}. Therefore, a new value for x is drawn from a proposal distribution $\textsc{Prop}(\Delta, v_{\text{PRE}})$ that depends on the previous value v_{PRE} for x.

The updated value β' of β is the usual update for an MH algorithm. As noted earlier, $\Delta_{\text{PRE}}(v_{\text{PRE}})$ is used for updating β and so, it is important to also keep track of the distribution from which a sample is drawn.

Finally, the state σ is updated to $\sigma[x \leftarrow v]$, the entry for x in the map Θ is appended with the value (v, Δ), and the tuple $(\sigma[x \leftarrow v], \beta', \textsc{Tail}(\Theta_{\text{PRE}}), \Theta(x)\text{++}[(v, \Delta)])$ is returned (where ++ denotes list concatenation).

It is also important to note that lines 8–14 in Algorithm 1 update β to take care of cases where the current value for a variable is undefined and the previous value is defined. This can be intuitively understood in a similar way as the update of β when a previous value is not available.

The following theorem states that for a probabilistic program $\mathcal{P}$, $\textsc{Infer}^{\checkmark}(\mathcal{P}, \kappa)$, computes answers that are consistent with $[\![\mathcal{P}]\!]$.

▶ **Theorem 4.1.** *For a program $\mathcal{P}$, the expectation of its return expression computed with respect to the samples generated by $\textsc{Infer}^{\checkmark}(\mathcal{P}, \kappa)$ approaches its denotational semantics $[\![\mathcal{P}]\!]$, as $\kappa \to \infty$.*

5 Evaluation

First, we show that $\textsc{Infer}^{\checkmark}$ computes correct answers for a set of micro-benchmark probabilistic programs. For these programs, other sampling based tools such as R2 and Stan compute answers that deviate significantly from the actual or analytically computed answers. Next, to demonstrate the practical applicability of $\textsc{Infer}^{\checkmark}$, we also show that it is able to work effectively for three real-world benchmark programs. All experiments were performed on a PC with a 2.00 GHz Intel 3rd Generation Core i7 processor and 8 GB RAM and running Microsoft Windows 8.1.

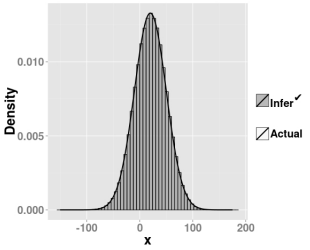
The first two micro-benchmarks that we consider are the programs in Figures 2(a) and 3(a). As can be seen in the plots 2(d) and 3(d), $\textsc{Infer}^{\checkmark}$ is able to estimate the correct answers. As discussed in Section 2, both R2 and Stan produce incorrect answers for these programs.

The other micro-benchmarks that we consider are shown in Figure 8. The benchmark (a) includes two assignments to the same variable `x`. The benchmark (b) is interesting because the variable `x` can be assigned different number of times across different runs of the program. If line 2 produces a value greater than 0.5, `x` is sampled again at line 4. On the other hand, if a sample value less than or equal to 0.5 is generated at line 2, then line 4 is not executed and `x` gets assigned only once. The benchmark (c) is a hierarchical model in which the prior distribution is estimated by means of a mixture of Gaussian distributions.

As seen from these figures, $\textsc{Infer}^{\checkmark}$ estimates distributions that coincide with the actual or analytically computed distributions. On the other hand, R2 and Stan produce incorrect results for all of the micro-benchmarks due to their lack of properly handling multiple sampling for the same variable and sampling from different execution paths.

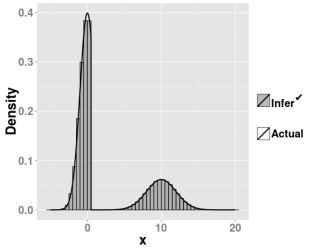
To demonstrate that $\textsc{Infer}^{\checkmark}$ is a practical algorithm, we also evaluate it on the following real-world benchmarks that are frequently used to test the robustness and scalability of Bayesian inference algorithms. As seen from the times reported below, $\textsc{Infer}^{\checkmark}$ is quite efficient on these benchmarks and therefore a practical solution.

```
1: double x;
2: x ~ Gaussian(10, 20);
3: x ~ Gaussian(20, 30);
4: return x;
```



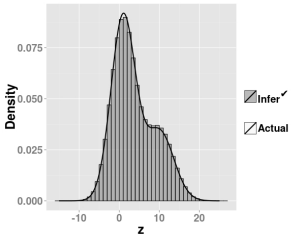
(a) Multiple assignments.

```
1: double x;
2: x ~ Gaussian(0, 1);
3: if (x > 0.5) then
4:   x ~ Gaussian(10, 2);
5: return x;
```



(b) Mixture model 1.

```
1: double x, y, z;
2: x ~ Gaussian(0, 1);
3: if (x > 0.5) then
4:   y ~ Gaussian(10, 2);
5: else
6:   y ~ Gamma(3, 3);
7: z ~ Gaussian(y, 3);
8: return z;
```



(c) Mixture model 2.

Figure 8 Micro-benchmarks.

- `Linear regression`: This is the standard Bayesian formulation of the linear regression model for fitting 1000 points [28] (*time taken by* Infer✓*: 12.55 seconds*).
- `HIV`: This is a multi-level or hierarchical linear model with varying slope and intercept. This model is for inferring the immunity levels in HIV-positive patients. The data comprises of 369 measurements taken over a two-year period on 84 patients [6] (*time taken by* Infer✓*: 4.36 seconds*).
- `Halo`: This is a skill rating system for a tournament of the Halo video game among 35 teams, with 2 players per team, and 500 games played between the teams [11] (*time taken by* Infer✓*: 5809 seconds*).

6 Related work

There has been significant progress in the development of probabilistic programming languages and tools in recent years [7, 21, 24, 8, 12, 22, 1, 9]. There are several approaches to performing inference for programs written in these languages: (1) by using static analysis techniques [20, 26, 4] such as abstract interpretation and data flow analysis, (2) by using dynamic analysis techniques [8, 12, 22, 2] such as MCMC sampling algorithms [19], or (3) by using Bayesian techniques where the program is compiled to a probabilistic model such as a Bayesian network [15] and inference is performed using probabilistic inference techniques such as belief propagation [23] over the probabilistic model [21].

We have observed that for techniques based on dynamic analysis, MH sampling based approaches in particular, tracking an ordered list of values that a variable gets assigned during a run of a program together with the distributions that were used to generate these values results in a correct sampling procedure for probabilistic programs. It might be tempting to consider a variable renaming scheme such as the static single assignment form (SSA) [5], or a variable indexing scheme [29] based on line numbers, distribution types, etc. (implemented in Stochastic Matlab). However, such schemes are inadequate to determine all values in the previous run that must be kept track of in order to propose a new value in the current run. This is clearly seen in Example 2 in Section 2.

For other inference techniques such as those based on static analysis [26] or Bayesian inference [21], it would be interesting to study analogous techniques for implementing provably correct inference algorithms.

7 Summary

We have highlighted the difficulties encountered in implementing a correct sampling-based inference engine for imperative probabilistic programs via several examples. We have designed an algorithm Infer$^{\checkmark}$ that overcomes these challenges, and generates samples from the correct distribution specified by the corresponding input probabilistic program.

Our algorithm is general and works for all probabilistic programs. We have also formally proved the correctness of our algorithm. We have implemented it in a tool called Infer$^{\checkmark}$, and have shown empirically that it works in all cases by comparing it with existing tools such as R2 and Stan. We have also shown that Infer$^{\checkmark}$ is a practical solution by evaluating it on real-world benchmarks.

Acknowledgements. We are grateful to Johannes Borgström for his valuable feedback on this paper.

References

1 Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. Measure transformer semantics for Bayesian machine learning. In *European Symposium on Programming (ESOP)*, pages 77–96, 2011.

2 Arun T. Chaganty, Aditya V. Nori, and Sriram K. Rajamani. Efficiently sampling probabilistic programs via program analysis. In *Artificial Intelligence and Statistics (AISTATS)*, 2013.

3 Siddhartha Chib and Edward Greenberg. Understanding the Metropolis-Hastings algorithm. *American Statistician*, 49(4):327–335, 1995.

4 Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. Bayesian inference for probabilistic programs via symbolic execution. In *Foundations of Software Engineering (FSE)*, 2013.

5 Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. An efficient method of computing static single assignment form. In *Principles of Programming Languages (POPL)*, pages 25–35, 1989.

6 Andrew Gelman and Jennifer Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2006.

7 W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

8 Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence (UAI)*, pages 220–229, 2008.

9 Andrew D. Gordon, Mihhail Aizatulin, Johannes Borgström, Guillaume Claret, Thore Graepel, Aditya V. Nori, Sriram K. Rajamani, and Claudio Russo. A model-learner pattern for Bayesian reasoning. In *Principles of Programming Languages (POPL)*, pages 403–416, 2013.

10 W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

11 Ralf Herbrich, Tom Minka, and Thore Graepel. TrueSkill: A Bayesian skill rating system. In *Neural Information Processing Systems (NIPS)*, pages 569–576, 2006.

12 Matthew D. Hoffman and Andrew Gelman. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, in press, 2013.

13 Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. Slicing probabilistic programs. In *Programming Languages Design and Implementation (PLDI)*, 2014.

14 S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI, 2007. `http://alchemy.cs.washington.edu`.

15 D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

16 D. Koller, D. A. McAllester, and A. Pfeffer. Effective Bayesian inference for stochastic programs. In *National Conference on Artificial Intelligence (AAAI)*, pages 740–747, 1997.

17 Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Science (JCSS)*, 22:328–350, 1981.

18 Jun S. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6(2):113–119, 1996.

19 David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

20 P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation. *Journal of Computer Security*, 2013.

21 Tom Minka, John Winn, John Guiver, and Anitha Kannan. Infer.NET 2.3, November 2009. Software available from `http://research.microsoft.com/infernet`.

22 Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. R2: An efficient MCMC sampler for probabilistic programs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

23 Judea Pearl. *Probabilistic Reasoning in Intelligence Systems*. Morgan Kaufmann, 1996.

24 Avi Pfeffer. The design and implementation of IBAL: A general-purpose probabilistic language. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

25 Walter Rudin. *Principles of mathematical analysis*. McGraw-Hill, 1976.

26 S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis of probabilistic programs: Inferring whole program properties from finitely many executions. In *Programming Languages Design and Implementation (PLDI)*, 2013.

27 Luke Tierney. Markov chains for exploring posterior distributions. *Annals of Statistics*, 22(4):1701–1728, 1994.

28 Gero Walter and Thomas Augustine. Bayesian linear regression - different conjugate models and their (insensitivity) to prior-data conflict. Technical report, University of Munich, TR-069, 2009.

29 David Wingate, Andreas Stuhlmüller, and Noah D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Artificial Intelligence and Statistics (AISTATS)*, 2011.

On the Problem of Computing the Probability of Regular Sets of Trees*

Henryk Michalewski[1] and Matteo Mio[2]

1 **University of Warsaw, Poland**
2 **CNRS/ENS-Lyon, France**

Abstract

We consider the problem of computing the probability of regular languages of infinite trees with respect to the natural coin-flipping measure. We propose an algorithm which computes the probability of languages recognizable by *game automata*. In particular this algorithm is applicable to all deterministic automata. We then use the algorithm to prove through examples three properties of measure: (1) there exist regular sets having irrational probability, (2) there exist comeager regular sets having probability 0 and (3) the probability of *game languages* $W_{i,k}$, from automata theory, is 0 if k is odd and is 1 otherwise.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases regular languages of trees, probability, meta-parity games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.489

1 Introduction

Regular languages of trees are sets of infinite binary trees, labeled by letters from a finite alphabet Σ, definable by a formula of Monadic Second Order (MSO) logic interpreted over the full binary tree [25] or, equivalently, specified by an *alternating tree automaton* [19].

In this paper we consider the following problem. Suppose a Σ-labeled tree t is generated by labeling each vertex by a randomly and uniformly chosen letter $a \in \Sigma$. For a given regular language L, what is the probability that t belongs to L? By probability we mean the standard *coin–flipping probability measure* μ (see Section 2 for definitions) on the space of Σ-labeled trees. Hence a precise formulation of our problem is as follows.

Probability Problem: Does there exist an algorithm which for a given regular language of trees L computes the probability $\mu(L)$?

A qualitative variant of the problem only asks for a decision procedure for the question "is $\mu(L) = 1$?". The problem is well posed since it was recently shown in [12, Theorem 1] that regular sets of trees are measurable with respect to any Borel measure and thus, in particular, with respect to the coin-flipping measure. An extended version of this paper is availiable at [15].

* The first author is supported by Polish National Science Centre grant no. 2014-13/B/ST6/03595. The second author is supported by the grant "Projet Émergent PMSO" of the École Normale Supérieure de Lyon.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 489–502

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Main Results

We give a positive solution to the Probability Problem for a subclass of regular languages.

▶ **Theorem 1.** *Let L be a regular set of infinite trees recognizable by a game automaton. Then the probability of L is computable and is an algebraic number.*

Game automata (see [10, 11, 21]) are special types of alternating parity tree automata. The class of languages recognizable by game automata includes, beside all *deterministic languages*, other important examples of regular sets. The most notable examples are the *game languages* $W_{i,k}$ which play a fundamental role in the study of tree languages with topological methods [3, 12]. Game automata definable languages are, at the present moment, the largest known subclass of regular languages for which the long-standing Mostowski–Rabin index problem[1] is known to be decidable (see [10, 11]). Theorem 1 confirms the good algorithmic properties of game automata. At the same time, however, we suspect that generalizing the result of Theorem 1 to arbitrary regular languages might be hard. Some ideas for further research in this direction are discussed in Section 6.

From Theorem 1 we derive the following propositions (for proofs see Section 5).

▶ **Proposition 2.** *There exists a regular language of trees L definable by a deterministic automaton such that L has an irrational probability.*

▶ **Proposition 3.** *There exists a regular language of trees L definable by a deterministic automaton such that L is comeager and L has probability* 0.

These two propositions should be contrasted with known properties of regular languages of infinite words. First, a result of Staiger in [22] states that a regular language L of infinite words has coin-flipping measure 0 if and only if it is of *Baire first category* (or *meager*). 3 shows that this correspondence fails in the context of infinite trees. Second, the coin-flipping measure of a regular language of infinite words is always rational (see, e.g., Theorem 2 of [7]). Hence, the probabilistic properties of regular languages of trees seem to be significantly more refined than in the case of languages of ω-words.

Lastly, we calculate the probability of all game languages $W_{i,k}$ (see [3, 12] and Subsection 5.4), a result that might eventually be useful given the importance of game languages in the topological study of regular sets of trees.

▶ **Proposition 4.** *For $0 \leq i < k$, the game language $W_{i,k}$ has probability* 0 *if k is odd and* 1 *if k is even.*

1.2 The Algorithm

In Section 4 we propose Algorithm 2 which computes the probability of regular languages recognized by game automata. Algorithm 2 is based on a reduction to *Markov Branching plays* (MBP's): to each game automaton $\mathcal{A}$ we associate a MBP $\mathcal{M}$. The *value* of $\mathcal{M}$ can be computed and corresponds to the probability of the language recognized by $\mathcal{A}$. This reduction to MBP's is described in Sections 3 and 4.

The notion of MBP, as a special kind of *two-player stochastic meta-parity game* has been introduced by the second author in [16, 17] in order to interpret a probabilistic version

[1] The Mostowski–Rabin Problem: for a given regular language L, compute the minimal number of *priorities* required to define L using an alternating parity tree automaton.

of the modal μ-calculus. For a given MBP $\mathcal{M}$ having n states, the vector $val \in [0,1]^n$ of values of $\mathcal{M}$ can be expressed as the solution of a system $\mathcal{S}$ of (nested) least and greatest fixed-point equations over the space $[0,1]^n$. From $\mathcal{S}$ one can then construct a first order formula $\phi_{\mathcal{S}}(val)$ in the language of real-closed fields having the property that val is the unique tuple of real numbers satisfying $\phi_{\mathcal{S}}$. The tuple val can be computed by Tarski's *quantifier elimination* algorithm [24] and consists of algebraic numbers. See Algorithm 1 in Section 3 for a description of the procedure for computing the value of MBP's.

One can find interesting the connections between the machinery of MBP's (and thus, as mentioned, the probabilistic μ-calculus), the class of languages definable by game automata, the algorithmic problem of computing the probability of regular languages of trees and the usage of Tarski's quantifier elimination procedure.

1.3 Related Work

In [23] L. Staiger presented an algorithm for computing the *Hausdorff measure* of regular sets of ω-words. The method, based on the decomposition of the input language into simpler components, can be adapted to compute the coin–flipping measure of regular sets of ω-words. Our research on the coin-flipping measure of regular languages of trees can be seen as a continuation of Staiger's work.

Natural variants of the qualitative version of the Probability Problem, obtained by replacing "has probability 1" by other notions of largeness, are known to have positive solutions: in [20] D. Niwiński described an algorithm which takes as input a regular language of trees L (presented as a Rabin tree automaton) and decides if L is uncountable and, similarly, an algorithm for establishing if a regular language of trees L is comeager can be extracted from the result of [14].

Addendum. After the submission of this article we have been informed that the Probability Problem has already been implicitly considered in [8], although differently phrased as the verification problem for a class of stochastic branching processes. Following our terminology, in [8] the authors provide an algorithm for computing the probability of regular languages definable by deterministic tree automata. Hence our results can be seen as extending the work of [8] from deterministic to game-automata definable languages.

2 Background in Topology and Automata Theory

2.1 Topology and measure

In this section we present elementary topological and measure–theoretical notions required in this work. We refer to [13] as a standard reference on the subject.

The set of natural numbers is denoted by ω. A topological space X is *Polish* if it is separable and completely metrizable. An important example of a Polish space is the *Cantor space* $\{0,1\}^\omega$ of infinite sequences of bits endowed with the product topology. In this paper we are interested in the probability Lebesgue measure μ on the product space Σ^I for I, a countable set of indices. The measure μ is uniquely defined by the assignment $\mu(\{t \in \Sigma^I \mid t(i_1) = a_1, \ldots, t(i_k) = a_k\}) = (\frac{1}{|\Sigma|})^k$ for $i_1, \ldots, i_k \in I$, $a_1, \ldots, a_k \in \Sigma$ ($i_j \neq i_{j'}$ whenever $j \neq j'$, see [13, Chapter 17] for additional details). In particular, for the alphabet $\Sigma = \{0,1\}$ and $I = \omega$ this is known as the coin–flipping probability measure on the Cantor space.

The countable set $V = \{L,R\}^*$ of finite words over the alphabet $\{L,R\}$ is called the *full binary tree* and each $v \in \{L,R\}^*$ is referred to as a *vertex*. The product space Σ^V is denoted

by $\mathcal{T}_\Sigma$ and an element $t \in \mathcal{T}_\Sigma$ is called a Σ-labeled tree, or just a Σ-tree. Intuitively, the stochastic processes associated with the coin–flipping measure μ on $\mathcal{T}_\Sigma$ generates an infinite Σ-tree by labeling each vertex with a randomly (uniformly) chosen label in Σ.

Given a topological space X, a set $A \subseteq X$ is *nowhere dense* if the interior of its closure is the empty set, that is $\mathtt{int}(\mathtt{cl}(A)) = \emptyset$. A set $A \subseteq X$ is of *(Baire) first category* (or *meager*) if A can be expressed as a countable union of nowhere dense sets. The complement of a meager set is called *comeager*.

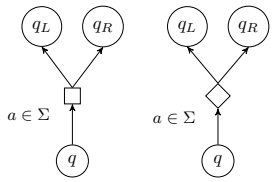
2.2 Alternating Parity Tree Automata and Game Automata

We include a brief exposition of alternating automata which follows the presentation in [19, Appendix C]. In this paper we are mostly interested in a subclass of alternating parity tree automata called game automata, which is introduced later in the Section.

▶ **Definition 5** (Alternating Parity Tree Automaton)**.** Given a finite set X, we denote with $\mathcal{DL}(X)$ the set of expressions e generated by the grammar $e ::= x \in X \mid e \wedge e \mid e \vee e$. An *alternating parity tree automaton* over a finite alphabet Σ is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \pi \rangle$ where Q is a finite *set of states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to \mathcal{DL}(\{L, R\} \times Q)$ is the *alternating transition function*, and $\pi : Q \to \omega$ is the *parity condition*.

An alternating parity tree automaton $\mathcal{A}$ over the alphabet Σ defines, or "accepts", a set of Σ-trees. The *acceptance* of a tree $t \in \mathcal{T}_\Sigma$ is defined via a two-player ($\exists$ and $\forall$) game of infinite duration denoted by $\mathcal{A}(t)$. Game states of $\mathcal{A}(t)$ are of the form $\langle \vec{x}, q \rangle$ or $\langle \vec{x}, e \rangle$ with $\vec{x} \in \{L, R\}^*$, $q \in Q$ and $e \in \mathcal{DL}(\{L, R\} \times Q)$.

The game $\mathcal{A}(t)$ starts at state $\langle \epsilon, q_0 \rangle$. Game states of the form $\langle \vec{x}, q \rangle$, including the initial state, have only one successor state, to which the game progresses automatically. The successor state is $\langle \vec{x}, e \rangle$ with $e = \delta(q, a)$, where $a = t(\vec{x})$ is the labeling of the vertex $\vec{x}$ given by t. The dynamics of the game at states $\langle \vec{x}, e \rangle$ depends on the possible shapes of e. If $e = e_1 \vee e_2$, then Player $\exists$ moves either to $\langle \vec{x}, e_1 \rangle$ or $\langle \vec{x}, e_2 \rangle$. If $e = e_1 \wedge e_2$, then Player $\forall$ moves either to $\langle \vec{x}, e_1 \rangle$ or $\langle \vec{x}, e_2 \rangle$. If $e = (L, q)$ then the game progresses automatically to the state $\langle \vec{x}.L, q \rangle$. Lastly, if $e = (R, q)$ the game progresses automatically to the state $\langle \vec{x}.R, q \rangle$. Thus a play in the game $\mathcal{A}(t)$ is a sequence Π of game–states, that looks like: $\Pi = (\langle \epsilon, q_0 \rangle, \ldots, \langle L, q_1 \rangle, \ldots, \langle LR, q_2 \rangle, \ldots, \langle LRL, q_3 \rangle, \ldots, \langle LRLL, q_4 \rangle, \ldots)$, where the dots represent part of the play in game–states of the form $\langle \vec{x}, e \rangle$. Let $\infty(\Pi)$ be the set of automata states $q \in Q$ occurring infinitely often in configurations $\langle \vec{x}, q \rangle$ of Π. We then say that the play Π of $\mathcal{A}(t)$ is winning for $\exists$, if $\max\{\pi(q) \mid q \in \infty(\Pi)\}$ is an even number. The play Π is winning for $\forall$ otherwise. The set (or "language") of Σ-trees defined by $\mathcal{A}$ is the collection $\{t \in \mathcal{T}_\Sigma \mid \exists \text{ has a winning strategy in the game } \mathcal{A}(t)\}$.



We reserve the symbols $\top$ and $\bot$ for two special *sink* states having even and odd priority, respectively. The transition function is defined, for all $a \in \Sigma$, as $\delta(\top, a) = (L, \top) \wedge (R, \top)$ and $\delta(\bot, a) = (L, \bot) \wedge (R, \bot)$. Clearly every tree is accepted at the state $\top$ and rejected at $\bot$. *Game automata* are a subfamily of alternating parity tree automata satisfying the constraint that, for each $q \in Q$ and $a \in \Sigma$, the transition $\delta(q, a) = e$ has either the form $e = (L, q_L) \vee (R, q_R)$ or $e = (L, q_L) \wedge (R, q_R)$ (see [10, 11] for more information about this class of automata). Transitions of a game automaton $\mathcal{A}$ can be schematically depicted as in the figure above with the left–hand and right–hand diagrams representing the transitions $(q, a) \to (L, q_L) \wedge (R, q_R)$ and $(q, a) \to (L, q_L) \vee (R, q_R)$, respectively. *Deterministic automata* are a subfamily of game automata satisfying the stronger constraint that, for each

$q \in Q$ and $a \in \Sigma$, the transition $\delta(q,a) = e$ has the form $e = (L, q_L) \wedge (R, q_R)$. Note that the sink states $\top$ and $\bot$ defined above have transitions satisfying this requirement.

3 Introduction to meta-parity games

In this Section we describe a class of stochastic processes called *Markov branching plays* (MBP's) [16, 17] which, as we will observe, is closely related to game automata and will provide a method for calculating the probability of regular languages defined by such automata. For a quick overview, a procedure for computing the value associated with a MBP is presented as Algorithm 1, at the end of this section. The procedure for computing the probability of regular languages defined by game automata in presented as Algorithm 2 in the next section.

We assume familiarity with the standard concepts of Markov chain and two-player stochastic ($2\frac{1}{2}$-player) parity game (see, e.g., [6]). Ordinary $2\frac{1}{2}$-player parity games are played on directed graphs whose set of states is partitioned into Player 1, Player 2 and probabilistic states. A $2\frac{1}{2}$-player parity game with neither Player 1 nor Player 2 states can be identified with a *Markov chain*.

Two-player stochastic meta-parity games [16, 17] generalize $2\frac{1}{2}$-player parity games by allowing the directed graph to have two additional kinds of states called $\exists$–branching states and $\forall$–branching states. In this paper we will only consider $2\frac{1}{2}$-player meta-parity games with neither Player 1 nor Player 2 states. Such structures, which thus constitute a generalization of Markov chains, are called *Markov branching plays* (MBP's). In what follows we provide a quick description of MBP and refer to [16] for a detailed account.

▶ **Definition 6** (Markov Branching Play)**.** A Markov branching play (MBP) is a structure $\mathcal{M} = \langle (S, E), (S_P, B_\exists, B_\forall), p, Par \rangle$ where:

- (S, E) is a directed graph with finite set of vertices S and transition relation E. We say that s' is a successor of s if $(s, s') \in E$. We assume that each vertex has at least one successor state in the graph (S, E).
- The triple $(S_P, B_\exists, B_\forall)$ is a partition of S into probabilistic, $\exists$-branching and $\forall$-branching states.
- The function $p : S_P \to (S \to [0,1])$ associates to each probabilistic state s a discrete probability distribution $p(s) : S \to [0,1]$ supported over the (nonempty) set of successors of s in the graph (S, E).
- Lastly, the function $Par : S \to \omega$ is the *parity (or priority) assignment*.

Recall that a Markov chain represents the stochastic process associated with a *random infinite walk* on its set of states. A MBP represents the more involved stochastic process, described below, of generation of a random unranked and unordered *tree* T whose vertices are labeled by states of the MDP.

MBP's as Stochastic Processes: given a MBP $\mathcal{M} = \langle (S, E), (S_P, B_\exists, B_\forall), p, Par \rangle$ and an initial vertex $s_0 \in S$, the stochastic process of construction of T is described as follows.

- The construction starts from the root of T which is labeled by s_0.
- A leaf x in the so far constructed tree T is extended, *independently* from all other leaves, depending on the type of its labeling state s, as follows:
  - If $s \in S_P$ then x is extended with a unique child which is labeled by a successor state s' of s randomly chosen in accordance with $p(s)$.
  - If $s \in B_\exists$ or $s \in B_\forall$ and $\{s_1, \ldots, s_n\}$ are the successors of s in $\mathcal{M}$, then x is extended with n children $y_1, \ldots y_n$ and y_i is labeled by s_i, for $1 \leq i \leq n$.

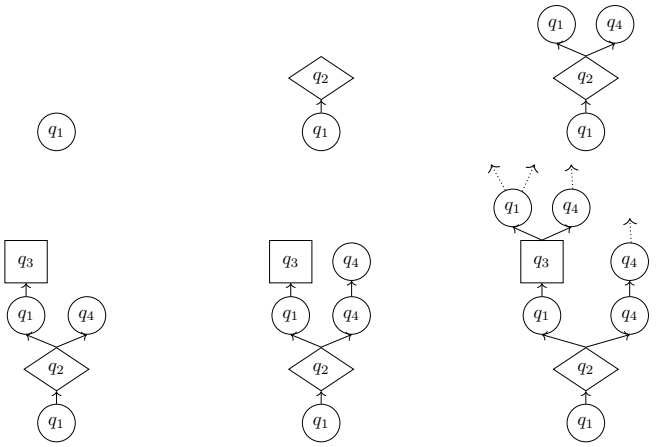


Figure 2 The stochastic process associated with the MBP in Figure 1.

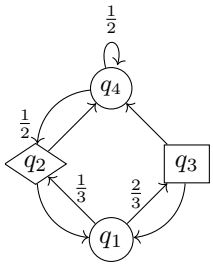


Figure 1 An example of a MBP.

We give in Figure 1 an example of a MBP. Probabilistic states, $\exists$-branching and $\forall$-branching states are marked as circles, diamond and boxes, respectively. The first six initial steps of the stochastic process associated with $\mathcal{M}$ at state q_1 are depicted in Figure 2. In the first step, the construction of T starts by labeling the root by q_1. Since q_1 is a probabilistic state, the tree is extended (second step) with only one child labeled by either q_2 (with probability $\frac{1}{3}$) or q_3 (prob. $\frac{2}{3}$). The picture shows the case when q_2 is chosen. Since the new leaf is labeled by q_2, and this is a $\exists$-branching state, the tree is extended by adding one new vertex for each successor of q_2 in $\mathcal{M}$, i.e., for both q_1 and q_4. The construction continues as described above. For example, the probability that the generated infinite tree will have the prefix as at the bottom right of Figure 2 is $\frac{1}{3} \cdot \frac{2}{3} \cdot \frac{1}{2} = \frac{2}{18}$.

The kind of infinite trees produced by the stochastic process just described are called *branching plays*. Branching plays are characterized by the property that each vertex labeled with a probabilistic state has only one child, and each vertex labeled with a ($\exists$ or $\forall$) branching state s has as many children as there are successors of s in the MBP.

The collection of branching plays in a MBP $\mathcal{M}$ starting from a state s is denoted by $\mathcal{BP}(\mathcal{M}, s)$. The set $\mathcal{BP}(\mathcal{M}, s)$ naturally carries a Polish topology making $\mathcal{BP}(\mathcal{M}, s)$ homeomorphic to the Cantor space (see, e.g., Definition 4.4 in [17]). The stochastic process associated to a MBP $\mathcal{M}$, specified on the previous page, can be naturally formalized by a probability measure $\mu_{\mathcal{M}}$ over the space $\mathcal{BP}(\mathcal{M}, s)$ of branching plays. See also Definition 4.7 in [17] for a formal definition.

Each branching play T can itself be viewed as an ordinary (infinite) two-player parity game $\mathcal{G}(T)$, played on the tree structure of T, by interpreting the vertices of T labeled by $\exists$-branching and $\forall$-branching states as under the control of Player $\exists$ and Player $\forall$, respectively. All other states (i.e., those labeled by a probabilistic state) have a unique successor in T to which the game $\mathcal{G}(T)$ progresses automatically. Lastly, the parity condition associated to each vertex corresponds to the parity assigned in $\mathcal{M}$ to the state labeling it. We denote with $\mathcal{W}_s$ the set of branching plays starting at s and winning for Player $\exists$, i.e., the set defined as: $\mathcal{W}_s = \{T \in \mathcal{BP}(\mathcal{M}, s) \mid \text{ Player } \exists \text{ has a winning strategy in } \mathcal{G}(T)\}$.

▶ **Definition 7** (Value of a MBP)**.** The *value* of a MBP $\mathcal{M}$ at a state s, denoted by $val(\mathcal{M}, s)$, is the probability of generating a branching play winning for $\exists$ starting the stochastic process from the state s. Formally, $val(\mathcal{M}, s) = \mu_{\mathcal{M}}(\mathcal{W}_s)$.

We remark that the above definition is valid because the set $\mathcal{W}_s$ is μ-measurable for every Borel measure μ on the space $\mathcal{BP}(\mathcal{M}, s)$ ([12]) and thus also for $\mu_{\mathcal{M}}$.

3.1 How to compute the value of a MBP

In this subsection we show how the values $val(\mathcal{M}, s)$ can be computed. The algorithm is based on a result of [16, 17], formulated as Theorem 10 below, characterizing such values as the solution of an appropriate system of (least and greatest) fixed-point equations. We first formulate Proposition 8 exposing a fixed-point property of the value of MBP's. Let us fix a MBP $\mathcal{M} = \langle (S, E), (S_P, B_\exists, B_\forall), p, Par \rangle$ with $S = \{s_1 \dots s_n\}$. To improve readability we just write val_i for $val(\mathcal{M}, s_i)$ and we denote with val the vector $val = (val_i)_{1 \le i \le n}$ of length n. The symbols $\sum$ and $\prod$ denote the usual operations of sum and product on reals. We also use a "coproduct" operation defined as $\coprod_{i \in I} x_i = 1 - \prod_{i \in I} 1 - x_i$.

▶ **Proposition 8.** *The equality $val = f(val)$ holds, where $f : [0,1]^n \to [0,1]^n$ is:*

$$\left(f \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}\right)_i = \begin{cases} \displaystyle\sum_{\{j \mid (s_i,s_j) \in E\}} p(s_i)(s_j) \cdot x_j & \text{if } s_i \in S_P \\ \displaystyle\prod_{\{j \mid (s_i,s_j) \in E\}} x_j & \text{if } s_i \in B_\forall \\ \displaystyle\coprod_{\{j \mid (s_i,s_j) \in E\}} x_j & \text{if } s_i \in B_\exists \end{cases}$$

Proof. Here we sketch the main idea of the argument, for a formal proof, see Theorem 4.22 of [17]. If s_i is a probabilistic state, then val_i is the weighted average of the value of its successors, since the stochastic process associated with the MBP chooses a unique successor s_j of s_i with probability $p(s_i)(s_j)$. If s_i is a $\forall$-branching state, then val_i is the probability that all *independently* generated subtrees are winning for Player $\exists$ and this is captured by the $\prod$ expression. Similarly, if s_i is a $\exists$-branching state then val_i is the probability that at least one generated subtree is winning for Player $\exists$, as formalized by the $\coprod$ expression. Hence the vector val is one of the fixed-points of the function $f : [0,1]^n \to [0,1]^n$. ◀

Theorem 10 below refines Proposition 8 by identifying val as the unique vector satisfying a system of nested (least and greatest) fixed-point equations. Its formulation closely follows the notation adopted in the textbook [1, §4.3] for presenting a similar result valid for ordinary parity games. To adhere to such notation, we will define a function g, a variant of the function f presented above. Let $k = \max\{Par(s) \mid s \in S\}$ and $l = \min\{Par(s) \mid s \in S\}$ be the maximal and minimal priorities used in the MBP, respectively, and let $c = k - l + 1$.

▶ **Definition 9.** The function $g : ([0,1]^n)^c \to [0,1]^n$ is defined as follows:

$$\left(g \begin{pmatrix} x_1^l \\ \vdots \\ x_n^l \end{pmatrix}, \dots, \begin{pmatrix} x_1^k \\ \vdots \\ x_n^k \end{pmatrix}\right)_i = \begin{cases} \displaystyle\sum_{\{j \mid (s_i,s_j) \in E\}} p(s_i)(s_j) \cdot x_j^{Par(s_j)} & \text{if } s_i \in S_P \\ \displaystyle\prod_{\{j \mid (s_i,s_j) \in E\}} x_j^{Par(s_j)} & \text{if } s_i \in B_\forall \\ \displaystyle\coprod_{\{j \mid (s_i,s_j) \in E\}} x_j^{Par(s_j)} & \text{if } s_i \in B_\exists \end{cases}$$

The function g depends, like the function f, only on n variables $\{x_1^{Par(s_1)}, \ldots, x_n^{Par(s_n)}\}$ appearing in the body of its definition. The input of g can indeed be regarded as the input of f divided in c baskets, where each variable x_i is put in the basket corresponding to the priority of s_i, for $1 \leq i \leq n$.

The set $[0,1]^n$, equipped with the pointwise order defined as $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n) \Leftrightarrow \forall i.(x_i \leq y_i)$, is a complete lattice and the function g is monotone with respect to this order in each of its arguments. Hence the Knaster–Tarski theorem ensures the existence of least and greatest points. We are now ready to state the main result regarding the values of a given MBP. We adopt standard μ-calculus notation (see, e.g., [1] and [16, 17]) to express systems of least and greatest fixed-points equations.

▶ **Theorem 10** ([16, Theorem 6.4.2]). *The following equality holds:*[2]

$$\begin{pmatrix} val_1 \\ \vdots \\ val_n \end{pmatrix} = \theta_k \begin{pmatrix} x_1^k \\ \vdots \\ x_n^k \end{pmatrix} . \cdots . \theta_l \begin{pmatrix} x_1^l \\ \vdots \\ x_n^l \end{pmatrix} . g(\begin{pmatrix} x_1^l \\ \vdots \\ x_n^l \end{pmatrix}, \ldots, \begin{pmatrix} x_1^k \\ \vdots \\ x_n^k \end{pmatrix})$$

where θ_i, for $l \leq i \leq k$ is a least-fixed point operator (μ) if i is an odd number and a greatest-fixed point operator if (ν) if i is even.

Proof. The proof goes by induction on the number of priorities in the MBP $\mathcal{M}$ and by transfinite induction on a rank-function defined on the space of branching plays. See [16] for a detailed proof. ◀

The next theorem states that the value of a MBP is computable and is always a vector of algebraic numbers. The examples discussed in Section 5 will illustrate the applicability of this result.

▶ **Theorem 11.** *Let $\mathcal{M}$ be a MBP. Then for each state s_i of $\mathcal{M}$ the value val_i is computable and is an algebraic number.*

Proof. (sketch) Using known ideas (see, e.g., Lemma 9 in [9] and Proposition 4.1 in [18]) the unique vector $val = (val_1, \ldots, val_n)$ satisfying the system of fixed-point expressions $\mathcal{S}$ given by Theorem 10 can be computed by a reduction to the first-order theory of real closed fields. A first order formula $F(x_1, \ldots, x_n)$, inductively defined from $\mathcal{S}$, is constructed with the property that $(val_1, \ldots, val_n) \in \mathbb{R}^n$ is the unique vector of reals satisfying the formula $F(x_1, \ldots, x_n)$. By Tarski's quantifier elimination procedure [24], the formula $F(x_1, \ldots, x_n)$ can be effectively reduced to an equivalent formula $G(x_1, \ldots, x_n)$ without quantifiers, that is, to a Boolean combination of equations and inequalities between polynomials over $(x_1, \ldots, x_n)$. It then follows that the $(val_1, \ldots, val_n)$, which can be extracted from G with standard methods, is a vector of algebraic numbers. In Section 5 we apply the above procedure to a number of examples. ◀

[2] Theorem 6.4.2 of [16] actually proves a stronger result valid for arbitrary $2\frac{1}{2}$-player meta-parity games whereas, as mentioned in the beginning of this section, Markov branching plays are $2\frac{1}{2}$-player meta-parity games without Player 1 and Player 2 states. Also, Theorem 6.4.2 of [16] is stated assuming the validity of the set-theoretic axiom $\mathrm{MA}_{\aleph_1}$, but as shown in [12] such assumption is not necessary and can thus be dropped.

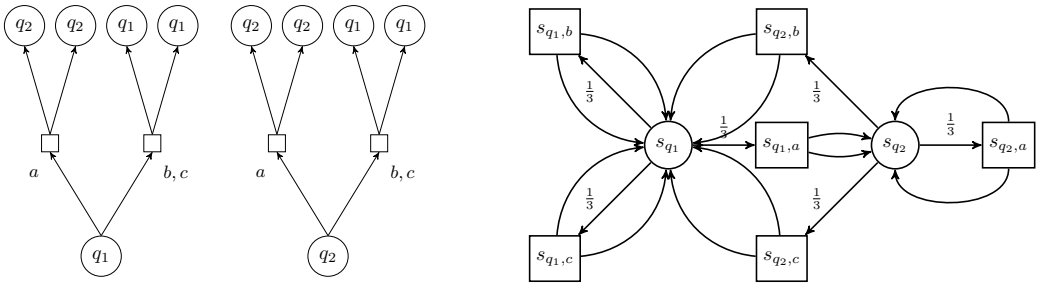


Figure 3 Transitions of the game automaton $\mathcal{A}$ and corresponding MBP $\mathcal{M}$.

1: ***input***: a Markov Branching Play $\mathcal{M}$.
output: algebraic numbers $r_1, \ldots, r_n \in \mathbb{R}$ equal to $(val_1, \ldots, val_n)$.
3: ***begin***
$\mathcal{S} \leftarrow$ Generate system of fixed–point equations associated to $\mathcal{M}$
5: $F(x_1, \ldots, x_n) \leftarrow$ Rewrite $\mathcal{S}$ to the corresponding first-order formula over $FO(\mathbb{R}, <, 0, 1, +, \times)$
$G(x_1, \ldots, x_n) \leftarrow$ Apply quantifier elimination procedure to $F(x_1, \ldots, x_n)$
7: ***return*** the unique vector $(r_1, \ldots, r_n)$ satisfying $G(x_1, \ldots, x_n)$

Algorithm 1: computing the vector of values of a MBP.

4 From Game Automata to Markov Branching Plays

In this section we present a reduction of the problem of computing the probability of regular languages definable by game automata to the problem of computing the value of a given MBP, which is algorithmically solvable using Algorithm 1.

We now describe how to construct from a game automaton $\mathcal{A} = (Q, q_0, \delta, \pi)$ over the alphabet Σ a corresponding MBP $\mathcal{M} = \langle (S, E), (S_P, B_\exists, B_\forall), p, Par \rangle$. The set S of states of $\mathcal{M}$ contains a probabilistic state s_q, for each $q \in Q$, a $\exists$–branching state $s_{q,a}$ for each pair (q, a), with $q \in Q$ and $a \in \Sigma$, such that $\delta(q, a) = (L, q_L) \vee (R, q_r)$, and a $\forall$–branching state $s_{q,a}$ for each pair (q, a) such that $\delta(q, a) = (L, q_L) \wedge (R, q_r)$. The transition relation E is defined as follows:

- a probabilistic state s_q has as successors the states $\{s_{q,a} \mid a \in \Sigma\}$,
- a $\exists$-branching (resp. $\forall$-branching) state $s_{q,a}$ have two successors s_{q_1} and s_{q_2} where $\delta(q, a) = (L, q_1) \vee (R, q_2)$ (resp. $\delta(q, a) = (L, q_1) \wedge (R, q_2)$).

Note that each state s_q, for $q \in Q$ has exactly $|\Sigma|$ successors and that each state $s_{q,a}$ has exactly[3] two successors. The assignment $p : S_P \to (S \to [0, 1])$ is defined as assigning to each probabilistic state (i.e., state of the form s_q) a uniform distribution over its successors, that is, $p(s_q)(s_{q,a}) = \frac{1}{|\Sigma|}$. Lastly, the parity assignment $Par : S \to \omega$ of the MBP $\mathcal{M}$ is defined as in the parity condition π of the game automaton $\mathcal{A}$ by the mapping $Par(s_q) = Par(s_{q,a}) = \pi(q)$.

As an illustrative example of this translation, consider the deterministic automaton $\mathcal{A} = \langle \{q_1, q_2\}, q_1, \delta, \pi \rangle$ over the alphabet $\Sigma = \{a, b, c\}$, with parity assignment $\pi(q_2) = 2$, $\pi(q_1) = 1$ and transition δ defined by $\delta(q_1, a) = \delta(q_2, a) = (L, q_2) \wedge (R, q_2)$ and $\delta(q_1, l) = \delta(q_2, l) = (L, q_1) \wedge (R, q_1)$, for $l \in \{a, b\}$.

[3] We are implicitly assuming, for the sake of simplicity, that each transition $(L, q_1) \wedge (R, q_2)$ and $(L, q_1) \vee (R, q_2)$ of δ in $\mathcal{A}$ is such that $q_1 \neq q_2$, and thus that $s_{q,a}$ has exactly two successors. If necessary, the game-automaton $\mathcal{A}$ can be made satisfy this assumption by introducing additional copies of the states.

The corresponding MBP $\mathcal{M}$ is schematically[4] depicted in Figure 3 (right), by representing probabilistic states with circles, $\forall$-branching states with boxes and the probabilistic assignment p by the probabilities labeling the outgoing edges of probabilistic states. The soundness of our reduction is stated as follows.

▶ **Theorem 12** (Correctness of Reduction). *Let L be a regular language recognized by a game automaton $\mathcal{A}$ and let $\mathcal{M}$ be the MBP corresponding to $\mathcal{A}$. Then $\mu(L) = Val(\mathcal{M}, s_{q_0})$, where q_0 is the initial state of $\mathcal{A}$.*

Proof. (sketch) Since each probabilistic state has exactly one successor for every letter $a \in \Sigma$ and each branching state have precisely two successors, there exists a one-to-one correspondence between Σ-trees $t \in \mathcal{T}_\Sigma$ and branching plays $T \in \mathcal{BP}(\mathcal{M}, s_{q_0})$. Furthermore, it follows directly from the definition of acceptance by $\mathcal{A}$ (see Section 2.2) and the definition of the set $\mathcal{W}_s$ (see Section 3) that t is accepted by $\mathcal{A}$ if and only if the corresponding branching play T is in $\mathcal{W}_s$. Lastly, due to the uniform assignment p of probabilities in $\mathcal{M}$, the coin-flipping measure μ on $\mathcal{T}_\Sigma$ and the probability measure $\mu_\mathcal{M}$ on $\mathcal{BP}(\mathcal{M}, s_{q_1})$ are identical. ◀

The result of Theorem 1 in the Introduction then follows as a corollary of Theorem 12 above and the fact that the vector of values of a MBP can be computed using Algorithm 1. The final algorithm for computing the probability of regular languages definable by game automata is then as follows.

```
1: input: a game automaton A=(Q, q0, δ, π) recognizing a language L.
   output: a real number corresponding to μ(L).
3: begin
     M ← Construct the MBP M corresponding to A
5:   (val_1, ..., val_n) ← Apply Algorithm 1 to compute the vector of values of the states of M
   return the value val_i where i is the index of the probabilistic state s_{q0} of M
```

Algorithm 2: computing the probability of regular languages L recognized by game automata.

5 Examples

In this section we will apply Algorithm 2 to analyze examples which will prove Propositions 2, 3 and 4 stated in the Introduction. In some instances, in order to perform the quantifier elimination procedure required by Algorithm 1, we use the tool `qepcad` [5].

We fix the alphabet $\Sigma = \{a, b, c\}$ and, for each $n \in \omega$, we define the regular language $L_n \subseteq \mathcal{T}_\Sigma$ as $L_n = \{t \in \mathcal{T}_\Sigma \mid a \text{ appears} \geq n \text{ times on every branch of } t\}$ and the language L_∞ as $L_\infty = \bigcap_{n \in \omega} L_n$, i.e., as the set of Σ-trees having, on every branch, infinitely many occurrences of the letter a.

5.1 An introductory example

The language L_1 is recognized by the deterministic automaton in Figure 4 (left) defined as $\mathcal{A}_1 = \langle \{q_1, \top\}, q_1, \delta_1, \pi \rangle$ where $\top$ is an accepting sink state (see Section 2.2 for automata–related definitions), the priority assignment is $\pi(q_1) = 1$ and the transition function δ_1 is defined on q_1 as $\delta_1(q_1, a) = (L, \top) \wedge (R, \top)$ and $\delta_1(q_1, l) = (L, q_1) \wedge (R, q_1)$ for $l \in \{b, c\}$.

[4] Due to the chosen succinct definition, the automaton $\mathcal{A}$ does not satisfy the assumption of Footnote 3. Rather than formally introducing copies q_1 and q_2 in $\mathcal{A}$, we have simply depicted all $\forall$-branching states of $\mathcal{M}$ as having two successors.

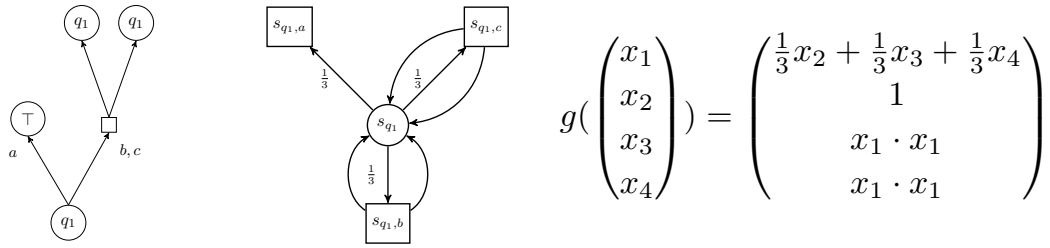


Figure 4 Automaton $\mathcal{A}_1$, MBP $\mathcal{M}_1$ and corresponding system of equations.

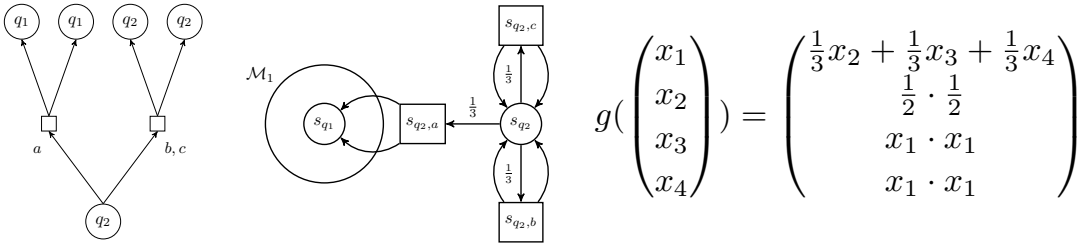


Figure 5 Automaton $\mathcal{A}_2$, MBP $\mathcal{M}_2$ and corresponding system of equations.

We will compute the probability $\mu(L_1)$ using the procedure of Algorithm 2. As a first step we construct the MBP $\mathcal{M}_1$ corresponding to $\mathcal{A}_1$, as specified in Section 4. In order to improve readability, we have represented in Figure 4 (center) a simplified version of $\mathcal{M}_1$ where the states $s_\top$, $s_{\top,a}$, $s_{\top,b}$ and $s_{\top,c}$ have been identified with the single state $s_{q_1,a}$. This is convenient since, clearly, all of these states have value 1. Accordingly, the MBP $\mathcal{M}_1$ has four states, all of priority 1. Following the procedure of Algorithm 2 we need to compute the values of the states of $\mathcal{M}_1$ using Algorithm 1. In accordance with Theorem 10, the fixed-point equation characterizing the vector $val = (val_{s_{q_1}}, val_{s_{q_1},a}, val_{s_{q_1},b}, val_{s_{q_1},c})$ of values of the states of $\mathcal{M}_1$ is $val = \mu\vec{x}.g(\vec{x})$, where g is defined as in Figure 4 (right). Then $val_{s_{q_1}}$ is the least solution in $[0,1]$ of the equation $x = \frac{1}{3} + \frac{2}{3}x^2$. As it is simple to verify, even without running the solver based on Tarski's quantifier elimination procedure, the solution is $val_{s_{q_1}} = \frac{1}{2}$, and this is the output returned by Algorithm 2. Hence the probability of L_1 is $\mu(L_1) = \frac{1}{2}$.

5.2 Examples of regular languages having irrational probabilities

This subsection constitutes a proof of Proposition 2. The automaton $\mathcal{A}_2$ recognizing the language L_2 is defined as $\mathcal{A}_2 = \langle(\{q_1, q_2, \top\}, q_2, \delta_2, \pi)$ where q_2 is the initial state, the priority function is defined as $\pi(q_1) = \pi(q_2) = 1$ and the transition function δ_2 is defined on q_1 as the function δ_1 of the previous example, and on the state q_2 as $\delta_2(q_2, a) = (L, q_1) \wedge (R, q_1)$ and $\delta_2(q_2, l) = (L, q_2) \wedge (R, q_2)$, for $l \in \{b, c\}$. The transition δ_2 is shown in Figure 5 (left).

The MBP $\mathcal{M}_2$ corresponding to $\mathcal{A}_2$ extends the MBP $\mathcal{M}_1$ of the previous example with the probabilistic state s_{q_2} and the three $\forall$-branching states $s_{q_2,a}$, $s_{q_2,b}$ and $s_{q_2,c}$. The new part of the automaton $\mathcal{A}_2$ is depicted in Figure 5 (center). Noting the four new states are not reachable by the other states already present in $\mathcal{M}_1$, we already know that $val_{s_{q_1}} = \frac{1}{2}$. Hence we can consider the simplified system of fixed-point equations $\mu\vec{x}.g(\vec{x})$ for calculating the values $val = (val_{q_2}, val_{q_2,a}, val_{q_2,b}, val_{q_2,c})$ where g is defined in Figure 5 (right). Hence the value val_{q_2} is the least solution in $[0,1]$ of the equation $x = \frac{1}{12} + \frac{2}{3}x^2$ and this is $val_{q_2} = \frac{1}{4}(3 - \sqrt{7})$ which is irrational and approximately equal to 0.088.

One can verify[5] that the probability of L_3 is $\mu(L_3) = \frac{1}{4}(3 - \sqrt{1+3\sqrt{7}})$ and thus not of the form $\frac{a+b\sqrt{c}}{d}$ for integers a, b, c, d. This means that $\mu(L_3)$ is not a quadratic irrational. By a characterization proved by Euler and Lagrange this in turn means that the continued fraction representation of $\mu(L_3)$ is not eventually periodic.

5.3 Example of a comeager language of probability 0

This subsection constitutes a proof of Proposition 3. The regular language L_∞ is recognized by the (deterministic) game automaton already defined in Section 4 and depicted in Figure 3 (left), where the states q_1 and q_2 have priority 1 and 2, respectively. The MBP associated with this automaton, depicted in Figure 3 (right), has eight states. The vector of values val is equal to $\nu\vec{y}^2.\mu\vec{y}^1.g(\vec{y}^1, \vec{y}^2)$ where

$$val = \begin{pmatrix} val_{s_{q_1}} \\ val_{s_{q_1,a}} \\ val_{s_{q_1,b}} \\ val_{s_{q_1,c}} \\ val_{s_{q_2}} \\ val_{s_{q_2,a}} \\ val_{s_{q_2,b}} \\ val_{s_{q_2,c}} \end{pmatrix} \quad \text{and} \quad g\left(\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \_ \\ \_ \\ \_ \\ \_ \end{pmatrix}, \begin{pmatrix} \_ \\ \_ \\ \_ \\ \_ \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{pmatrix}\right) = \begin{pmatrix} \frac{1}{3}y_2 + \frac{1}{3}y_2 + \frac{1}{3}y_4 \\ y_5 \cdot y_5 \\ y_1 \cdot y_1 \\ y_1 \cdot y_1 \\ \frac{1}{3}y_5 + \frac{1}{3}y_1 + \frac{1}{3}y_1 \\ y_5 \cdot y_5 \\ y_1 \cdot y_1 \\ y_1 \cdot y_1 \end{pmatrix}$$

By straightforward simplifications we obtain the system of fixed-point equations

$$\begin{cases} x_1 \overset{\mu}{=} \frac{1}{3}x_2^2 + \frac{2}{3}x_1^2 \\ x_2 \overset{\nu}{=} \frac{1}{3}x_2^2 + \frac{2}{3}x_1^2 \end{cases}$$

in the two variables x_1 and x_2 (corresponding to the variables y_1, representing s_{q_1}, and y_5, representing s_{q_2}). The execution[6] of Algorithm 1 reveals that the solution of the system of equations is $(0, 0)$. Hence $val_{s_{q_2}} = 0$ and this shows that the probability $\mu(L_\infty)$ of the language L_∞ is 0.

5.4 Computing the measure of $W_{i,k}$

The family of regular languages $W_{i,k}$, indexed by pairs $i < k$ of natural numbers, constitutes a tool for investigating properties of regular languages using topological methods ([2, p. 329], see also [3, 4, 12]). The standard game automaton $\mathcal{A}_{i,k}$ over the language $\Sigma_{i,k} = \{\forall, \exists\} \times \{i, i+1, \ldots, k-1, k\}$ accepting $W_{i,k} \subseteq \mathcal{T}_{\Sigma_{i,k}}$ is defined as $\mathcal{A}_{i,k} = \langle Q, q_i, \delta, \pi \rangle$ where $Q = \{q_i, q_{i+1}, \ldots, q_k\}$, the initial state is q_i and, for each $i \leq j \leq k$, the state q_j has priority $\pi(q_j) = j$ and the transition function δ is defined on q_j as in Figure 6. Our proof of Proposition 4, stated in the Introduction, goes by analyzing the system of fixed-point

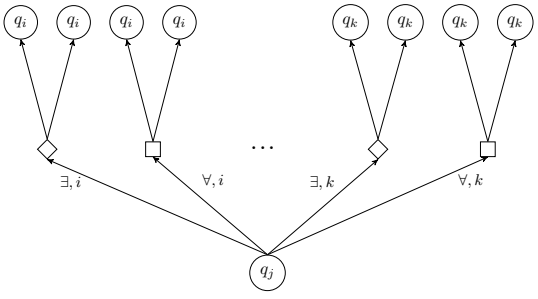


Figure 6 Transition of $\mathcal{A}_{i,k}$ recognizing $W_{i,k}$.

[5] See Section 5 of [15].

[6] Details are presented in Section 5 of [15] along with a proof that the set $L_\infty \subseteq \mathcal{T}_{a,b,c}$ is comeager.

equations associated with the game automaton $\mathcal{A}_{i,k}$. Importantly, such a system consists of linear equations and not, as in the general case, of higher order polynomials. This system can be solved using standard techniques of linear algebra. A detailed proof of Proposition 4 can be found in Subsection 5.6 of [15].

6 Conclusion

In this work we presented an algorithm for computing the probability of regular languages defined by game automata. The Probability Problem in its full generality remains open. A possible direction for future research is to investigate approximations of regular languages by simpler regular languages. For example, given a regular language L of trees, is it possible to find a regular language G defined by a game automaton such that $L \triangle G = (L \setminus G) \cup (G \setminus L)$ is of probability 0, i. e. L differs from G by a set of probability 0? An effective answer to this question, that is an algorithm constructing a language G from L, combined with the algorithm described in this paper would lead to a full solution to the Probability Problem.

References

1 A. Arnold and D. Niwiński. *Rudiments of μ-calculus.* Studies in Logic. North-Holland, 2001.

2 André Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.

3 André Arnold and Damian Niwiński. Continuous separation of game languages. *Fundamenta Informaticae*, 81:19–28, 2008.

4 Julian C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theor. Comput. Sci.*, 195(2):133–153, 1998.

5 Christopher W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using cads. *SIGSAM Bull.*, 37(4):97–108, 2003.

6 Krishnendu Chatterjee. *Stochastic ω-Regular Games.* PhD thesis, University of California, Berkeley, 2007.

7 Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Quantitative stochastic parity games. In *Proc. of SODA*, pages 121–130, 2004.

8 Taolue Chen, Klaus Dräger, and Stefan Kiefer. Model checking stochastic branching processes. In *Proceedings of MFCS*, volume 7468 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2012.

9 Luca de Alfaro and Rupak Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.

10 Jacques Duparc, Alessandro Facchini, and Filip Murlak. Definable operations on weakly recognizable sets of trees. In *Proc. of FSTTCS*, pages 363–374, 2011.

11 Alessandro Facchini, Filip Murlak, and Michal Skrzypczak. Rabin-Mostowski index problem: A step beyond deterministic automata. In *Proc. of LICS*, pages 499–508, 2013.

12 Tomasz Gogacz, Henryk Michalewski, Matteo Mio, and Michal Skrzypczak. Measure properties of game tree languages. In *Proc. MFCS*, pages 303–314, 2014.

13 A. S. Kechris. *Classical Descriptive Set Theory.* Springer Verlag, 1994.

14 Henryk Michalewski and Matteo Mio. Baire Category Quantifier in Monadic Second Order Logic. In *Proc. of ICALP*, 2015.

15 Henryk Michalewski and Matteo Mio. On the problem of computing the probability of regular sets of trees. *CoRR*, abs/1510.01640, 2015.

16 Matteo Mio. *Game Semantics for Probabilistic μ-Calculi.* PhD thesis, School of Informatics, University of Edinburgh, 2012.

17 Matteo Mio. Probabilistic Modal μ-Calculus with Independent product. *Logical Methods in Computer Science*, 8(4), 2012.

18 Matteo Mio and Alex Simpson. Łukasiewicz mu-calculus. In *Proc. of Workshop on Fixed Points in Computer Science*, volume 126 of *EPTCS*, 2013.

19 David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

20 Damian Niwiński. On the cardinality of sets of infinite trees recognizable by finite automata. In *Proc. MFCS*, 1991.

21 Damian Niwiński and Igor Walukiewicz. A gap property of deterministic tree languages. *Theor. Comput. Sci.*, 1(303):215–231, 2003.

22 Ludwig Staiger. Rich omega-words and monadic second-order arithmetic. In *Proc. of CSL*, pages 478–490, 1997.

23 Ludwig Staiger. The Hausdorff measure of regular omega-languages is computable. *Bulletin of the EATCS*, 66:178–182, 1998.

24 Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951.

25 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.

Probabilistic Regular Expressions and MSO Logic on Finite Trees

Thomas Weidner*

Leipzig University, Institute of Computer Science, Leipzig, Germany
weidner@informatik.uni-leipzig.de

Abstract

We introduce probabilistic regular tree expressions and give a Kleene-like theorem for probabilistic tree automata (PTA). Furthermore, we define probabilistic MSO logic. This logic is more expressive than PTA. We define bottom-up PTA, which are strictly more expressive than PTA. Using bottom-up PTA, we prove a Büchi-like theorem for probabilistic MSO logic. We obtain a Nivat-style theorem as an additional result.

1998 ACM Subject Classification F.1.1 Models of Computation, G.3 Probability and Statictics

Keywords and phrases Probabilistic Regular Expressions, MSO Logic, Tree Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.503

1 Introduction

Probabilistic tree automata (PTA) were introduced by Magidor [13] and Ellis [11] in the 1970s. These automata enjoy plentiful applications in the field of natural language processing, including parsing, deep language models, and machine translation. We consider the behaviour of a PTA as a function mapping a tree to a probability value. Recent research has already transferred the Kleene- and Büchi theorems on words to the probabilistic setting [4, 14, 17, 18] as well as to the weighted setting [7, 9]. The classical Nivat theorem [16] characterises regular tree transductions by decompositions in a regular tree language and homomorphisms. Nivat characterisations have attracted recent interest [1, 8]. In this work, we present probabilistic variants of these classical results for finite trees.

We introduce probabilistic regular tree expressions (PRTE). Compared to the existing regular tree expressions we use a different iteration operator $S^{\infty z}$, which we call infinity iteration. The usual Kleene-iteration involves a choice after every iteration step either to stop or to continue the iteration. Our iteration removes this ambiguity and forces the iteration to continue until there are no more variables to substitute.

In order to obtain a probabilistic extension of MSO logic, we add a second order expected value operator $\mathbb{E}_p X.\varphi$ to MSO logic. In the scope of this operator, formulas $x \in X$ are considered to be true with probability p. The semantics of the expected value operator is then defined as the expected value over all sets. It turns out that standard (top-down) PTA are not expressive enough to capture the semantics of this probabilistic MSO logic. Therefore, we introduce bottom-up PTA. These automata assign a probability to a state given all the states at the child nodes. Thus, bottom-up PTA are a generalisation of deterministic bottom-up tree automata, and are strictly more expressive than (top-down) PTA.

The main results of this paper are the following:

1. We prove that PRTE and top-down PTA are expressively equivalent.

* Partially supported by DFG Graduiertenkolleg 1763 (QuantLA).

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 503–516

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2. We prove a Nivat-theorem, which states that the behaviours of PTA are exactly the functions which can be constructed from regular tree languages using operations like relabellings, intersections, and probability measures.
3. Using this Nivat-theorem, we show the expressive equivalence of probabilistic MSO logic and bottom-up PTA.

For the first result, we employ so-called substitution summable PTA, where variables can only be accepted in sink states. The use of our Nivat-theorem for the third result allows us to reuse the classical Büchi-theorem.

2 Preliminaries

Let $\mathbb{N}$ be the set $\{1, 2, 3, \ldots\}$ and $\mathbb{N}_0$ be $\mathbb{N} \cup \{0\}$. Any finite, non-empty set Σ is called an alphabet. By Σ^* we denote all words over Σ and by Σ^+ all words over Σ excluding the empty word ε.

A *rank alphabet* is a finite, non-empty set Σ with a function $ar\colon \Sigma \to \mathbb{N}_0$ which assigns to every symbol $f \in \Sigma$ its arity. For convenience we let $\Sigma_n = \{ f \in \Sigma \mid ar(f) = n \}$ for every $n \in \mathbb{N}_0$. We write Σ for (Σ, ar) if ar is understood.

A *tree over* Σ is a function $t\colon D \to \Sigma$ where $D \subseteq \mathbb{N}^*$ is a non-empty, finite, prefix-closed set such that $\{ i \in \mathbb{N} \mid xi \in D \} = \{1, \ldots, ar(t(x))\}$ for every $x \in D$. We write $\mathrm{dom}(t)$ for D and $\mathrm{dom}_A(t)$ for all $x \in \mathrm{dom}(t)$ with $t(x) \in A$. We denote by $\mathrm{leaf}(t)$ the set of all maximal positions with respect to the prefix order, and by $\mathrm{inner}(t)$ the set $\mathrm{dom}(t) \setminus \mathrm{leaf}(t)$. The set of all trees over Σ is written T_Σ.

Let $t \in \mathrm{T}_\Sigma$ and $x \in \mathrm{dom}(t)$. We write $t|_x$ for the *subtree of t rooted at x* given by $\mathrm{dom}(t|_x) = \{ y \mid xy \in \mathrm{dom}(t) \}$ and $t|_x(y) = t(xy)$. For $f \in \Sigma_n$ and $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$ let $f(t_1, \ldots, t_n)$ be the tree $\bar{t}$ given by $\mathrm{dom}(\bar{t}) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} i\,\mathrm{dom}(t_i)$, $\bar{t}(\varepsilon) = f$ and $\bar{t}(ix) = t_i(x)$ for $i \in \{1, \ldots, n\}$ and $x \in \mathrm{dom}(t_i)$. For an introduction into tree automata, regular tree expressions, and logic on finite trees, see [6].

By $\mathbb{1}_Y\colon X \to \{0, 1\}$ we denote the characteristic function of $Y \subseteq X$. For a countable, non-empty set X let $\Delta(X)$ denote the set of all *distributions on X*, i.e. all functions $d\colon X \to [0, 1]$ such that $\sum_{x \in X} d(x) = 1$. Let $\Delta_0(X) = \Delta(X) \cup \{\mathbb{0}_X\}$, where $\mathbb{0}_X$ is the functions which assigns 0 to every $x \in X$. We call any function $S\colon \mathrm{T}_\Sigma \to [0, 1]$ a probabilistic tree series or just a tree series.

▶ **Definition 1.** Let Σ be a rank alphabet. A *(top-down) probabilistic tree automaton (PTA)* is a quadruple $A = (Q, \delta, \mu, F)$ where

1. Q is a finite, non-empty set – the set of states,
2. $\delta = \bigcup_{n \geq 1} \delta_n$ where $\delta_n\colon Q \times \Sigma_n \to \Delta_0(Q^n)$ – the transition probability function
3. $\mu \in \Delta(Q)$ – the initial distribution,
4. $F \subseteq Q \times \Sigma_0$ – the acceptance condition.

The *behaviour of A* is a function $\|A\|\colon \mathrm{T}_\Sigma \to [0, 1]$ defined by

$$\|A\|(t) = \sum_{\substack{\rho\colon \mathrm{dom}(t) \to Q \\ (\rho(x), t(x)) \in F \text{ for all } x \in \mathrm{leaf}(t)}} \mu(\rho(\varepsilon)) \prod_{x \in \mathrm{inner}(t)} \delta(\rho(x), t(x))\big(\rho(x1), \ldots, \rho(x\,ar(t(x)))\big).$$

A state $q \in Q$ is called a *sink* if $\delta(q, f) = \mathbb{0}$ for all $f \in \Sigma$.

3 Probabilistic Regular Tree Expressions

Like with classical regular tree expressions, we introduce an additional set of variables which will be used to mark the positions where tree substitution can occur. Formally for a finite set V let $\mathrm{T}_\Sigma(V) = \mathrm{T}_{\Sigma'}$ where $\Sigma'_n = \Sigma_n$ for $n \geq 1$ and $\Sigma'_0 = \Sigma_0 \cup V$. The special notation for trees containing variables is used to emphasize the special role of variables.

3.1 Operations on Tree Series

Before we define the syntax and semantics of probabilistic regular tree expressions, we introduce the operations used in these expressions.

For two trees $s, t \in \mathrm{T}_\Sigma(V)$ and a set of variables $W \subseteq V$, we define $s \trianglelefteq_W t$ to hold if and only if t can be obtained by substituting all variables from W in s, i.e, $\mathrm{dom}(s) \subseteq \mathrm{dom}(t)$ and $s(x) = t(x)$ for all $x \in \mathrm{dom}_{(\Sigma \cup V) \setminus W}(s)$. Then, $\trianglelefteq_W$ is a quasi-order, which we call the *substitution order*. For $|W| = 1$ we even have that $\trianglelefteq_W$ is a partial order, i.e, it is also anti-symmetric.

▶ **Definition 2.** Let $S, T \colon \mathrm{T}_\Sigma(V) \to [0,1]$ and $z \in V$. We define the *concatenation* $S \cdot_z T$ of S and T by

$$\big(S \cdot_z T\big)(t) = \sum_{s \trianglelefteq_z t} S(s) \cdot \prod_{x \in \mathrm{dom}_z(s)} T(t|_x) \qquad \text{for all } t \in \mathrm{T}_\Sigma(V).$$

Note that this definition is the same as for weighted tree series as given in [7]. They also showed that this product is associative for a fixed variable z, i.e., $(R \cdot_z S) \cdot_z T = R \cdot_z (S \cdot_z T)$. This does not hold if two different variables are used.

It is easy to see, that $S \cdot_z T$ can assume values outside of $[0,1]$ if we allow arbitrary tree series S and T. Hence, we make the assumption that for any given tree t the tree series S is a distribution on the trees s which can be extended using tree substitution to obtain t. More formally:

▶ **Definition 3.** A tree series $S \colon \mathrm{T}_\Sigma(V) \to [0,1]$ is called *substitution summable* if

$$\sum_{s \trianglelefteq_V t} S(s) \leq 1$$

holds for all $t \in \mathrm{T}_\Sigma(V)$.

Restricting S to be substitution summable in Definition 2 assures that $S \cdot_z T$ is bounded by 1. In addition, substitution summability is preserved by $\cdot_z$.

▶ **Lemma 4.** *Let $S, T \colon \mathrm{T}_\Sigma(V) \to [0,1]$ and $z \in V$. If S is substitution summable, then $(S \cdot_z T)(t) \leq 1$ for all $t \in \mathrm{T}_\Sigma(V)$. Moreover, if T is also substitution summable, so is $S \cdot_z T$.*

Proof. The first claim is easy to see as the set of all s with $s \trianglelefteq_V t$ contains all trees s with $s \trianglelefteq_z t$. For the second statement let $t \in \mathrm{T}_\Sigma(V)$. We compute

$$\sum_{s \trianglelefteq_V t} \sum_{r \trianglelefteq_z s} S(r) \prod_{x \in \mathrm{dom}_z(r)} T(s|_x) = \sum_{r \trianglelefteq_V t} S(r) \prod_{x \in \mathrm{dom}_z(r)} \sum_{s_x \trianglelefteq_V t|_x} T(s_x) \leq 1.$$

Here, we applied the index transformation $(s, r) \mapsto (r, (s|_x)_{x \in \mathrm{dom}_z(r)})$, which is bijective map from $\{(s,r) \mid r \trianglelefteq_z s \trianglelefteq_V t\}$ to $\{(r, (s_x)_{x \in \mathrm{dom}_z(r)}) \mid r \trianglelefteq_V t \text{ and } s_x \trianglelefteq_V t|_x \text{ for all } x \in \mathrm{dom}_z(r)\}$. ◀

Next, we give a probabilistic iteration operation. The usual Kleene-iteration adds a choice after every step to either stop the iteration process or substitute the variable again. This non-deterministic choice cannot be easily modelled probabilistically. Therefore, we propose a slightly different notion of iteration, where this choice is not present.

▶ **Definition 5.** Let $S\colon \mathrm{T}_\Sigma(V) \to [0,1]$ and $z \in V$. We define the *infinity iteration* $S^{\infty z}$ by

$$S^{\infty z}(t) = \lim_{n\to\infty} S^{\cdot z n}(t)$$

for all $t \in \mathrm{T}_\Sigma$, where $S^{\cdot z 0} = \mathbb{1}_{\{z\}}$ and $S^{\cdot z n+1} = S \cdot_z S^{\cdot z n}$ for all $n \geq 0$.

▶ **Lemma 6.** *Let $S\colon \mathrm{T}_\Sigma(V) \to [0,1]$ substitution summable and $z \in V$. Then $S^{\infty z}(t)$ is well-defined for all $t \in \mathrm{T}_\Sigma(V)$, i.e., the limit always converges and attains values in $[0,1]$, and is again substitution summable.*

Proof. First consider the case that $z \notin t(\mathrm{dom}(t))$. We then have $S^{\cdot z (n+1)}(t) \geq S^{\cdot z n}(t)$ by the definition of tree series concatenation. Thus, the sequence $S^{\cdot z n}(t)$ is monotonically increasing and bounded by Lemma 4. Hence, the sequence converges.

Now let $z \in t(\mathrm{dom}(t))$. We may assume that $S(z) < 1$ as otherwise S would be equal to $\mathbb{1}_{\{z\}}$. Note that for any trees s', s with $s' \trianglelefteq_z s$ it holds that whenever $z \in s(\mathrm{dom}(s))$ also $z \in s'(\mathrm{dom}(s'))$. Hence, we obtain

$$S^{\cdot z n}(t) = \sum_{t_0 \trianglelefteq_z \cdots \trianglelefteq_z t_n = t} S(t_0) \prod_{i=1}^{n} \prod_{x \in \mathrm{dom}_z(t_{i-1})} S(t_i|_x) \leq \sum_{k=0}^{n} \sum_{\substack{t_0 \trianglelefteq_z \cdots \trianglelefteq_z t_n = t \\ |\{\, i \mid t_{i-1} \neq t_i \,\}| = k}} S(z)^{n-k}.$$

Let N be the finite number of trees s with $s \trianglelefteq_z t$. Thus, we can bound k in the above equation by N. Any chain $t_0 \trianglelefteq_z \cdots \trianglelefteq_z t_n = t$ can be uniquely identified by the choice of k positions where inequality occurs and the trees occurring at these positions. Hence, there are at most $\sum_{k=0}^{N} \binom{n}{k} N^k$ chains of length n. Therefore, $S^{\cdot z n}(t) \leq P(n) S(z)^{n-N}$ for some polynomial P of degree independent of n. Thus, $S^{\cdot z n}(t) \to 0$ as $n \to \infty$ and so $S^{\infty z}(t) = 0$.

By Lemma 4 we know that $S^{\cdot z n}$ is substitution summable for any $n \geq 1$. Let $t \in \mathrm{T}_\Sigma(V)$. We obtain

$$\sum_{s \trianglelefteq_V t} S^{\infty z}(s) = \sum_{s \trianglelefteq_V t} \lim_{n\to\infty} S^{\cdot z n}(s) = \lim_{n\to\infty} \sum_{s \trianglelefteq_V t} S^{\cdot z n}(s) \leq 1. \qquad \blacktriangleleft$$

▶ Remark. In [7] an alternative iteration for tree series was proposed: Suppose that $S\colon \mathrm{T}_\Sigma(V) \to \mathbb{K}$ is a weighted tree series, where $\mathbb{K}$ is a semiring, with $S(z) = 0_\mathbb{K}$. They define $S_z^{0,\mathrm{F}} = \mathbb{0}$, $S_z^{n+1,\mathrm{F}} = S \cdot_z (S_z^{n,\mathrm{F}} + \mathbb{1}_{\{z\}})$, and $S_z^{*,\mathrm{F}}(t) = S_z^{\mathrm{height}(t)+1,\mathrm{F}}(t)$. As can be seen from the definition of $S_z^{n+1,\mathrm{F}}$, there is a choice for every leaf labelled by z to either continue the iteration or to just stop and attach the weight one to this position. There is no such choice with $S^{\infty z}$ – the iteration always has to continue. Nevertheless, when no leaf is labelled z, the iteration in $S_z^{*,\mathrm{F}}$ cannot stop at a z labelled leaf and thus equals to $S^{\infty z}$, i.e., $S^{\infty z}(t) = S_z^{*,\mathrm{F}}(t)$ for all $t \in \mathrm{T}_\Sigma(V)$ with $z \notin t(\mathrm{dom}(t))$ if $S(z) = 0$ and S substitution summable.

The usual Kleene-iteration $L^{*,z}$ of a tree language L with $z \notin L$ can be characterised as the unique solution of the equation $X = L \cdot_z X \cup \{z\}$. An analogous fixed-point characterisation can also be given for $S^{\infty z}$:

▶ **Corollary 7.** *Let $z \in V$ and S be a substitution summable probabilistic tree series with $S(z) < 1$. Then, $S^{\infty z}$ is the unique solution of the equation $X = S \cdot_z X$.*

Proof. By Lemma 6, $S^{\infty z} = \lim_{n\to\infty} S^{\cdot_z n}$ is well-defined. Thus, $S^{\infty z}$ is a solution of $X = S \cdot_z X$. Consider a tree series T with $T = S \cdot_z T$, i.e., $T = S^{\cdot_z n} \cdot_z T$ for all $n \geq 0$. We obtain

$$\begin{aligned} T(t) &= \lim_{n\to\infty} \big(S^{\cdot_z n} \cdot_z T\big)(t) = \lim_{n\to\infty} \sum_{s \trianglelefteq_z t} S^{\cdot_z n}(s) \prod_{x \in \mathrm{dom}_z(s)} T(t|_x) \\ &= \sum_{s \trianglelefteq_z t} \Big(\lim_{n\to\infty} S^{\cdot_z n}(s)\Big) \prod_{x \in \mathrm{dom}_z(s)} T(t|_x) = (S^{\infty z} \cdot_z T)(t) = S^{\infty z}(t), \end{aligned}$$

where the last equality holds as $S^{\infty z} \cdot_z T = S^{\infty z}$. This is due to the proof of Lemma 6, where we established that $S^{\infty z}(t) = 0$ if t contains the symbol z at any leaf. ◀

3.2 Syntax and Semantics of Regular Expressions

The syntax of regular tree expressions is based on weighted regular tree expressions with two differences: First, we use infinity-iteration instead of Kleene-iteration, and second, we do not allow arbitrary summation of terms. Instead, we use two restricted rules for sums, which either decide based on the root symbol or model probabilistic branching. Furthermore, we do not allow (direct) summation of variables. Unfortunately, these restrictions remove the closure of the set of expressions under associativity, commutativity and distributivity. Hence, we explicitly add these rules to the syntax. Though these rules are not needed to add expressiveness, we include the rules nevertheless to make it possible to write more natural expressions. This is formalised below.

▶ **Definition 8.** The set pRTE of *probabilistic regular tree expressions* is the smallest set $\mathcal{E}$ that is closed under the following grammar rules (where $p \in [0,1]$, $\Sigma' \subseteq \Sigma$ and $z \in V$)

$$E ::= \mathbb{0} \mid z \mid \sum_{f\in\Sigma'} f(E,\dots,E) \mid pE + (1-p)E \mid E \cdot_z E \mid E^{\infty z},$$

and is also closed under the following identities. Each identity states that an expression containing the left side of an identity as a subexpression is in $\mathcal{E}$ if and only if the same expression, but with this subexpression replaced by the right side of the identity, is in $\mathcal{E}$. The following identities model associativity of $\cdot$, $+$ and $\cdot_z$, commutativity of $+$:

$$\begin{aligned} (E+F)+G &= E+(F+G), & E+F &= F+E, \\ (E \cdot_z F) \cdot_z G &= E \cdot_z (F \cdot_z G), & (p_1p_2)E &= p_1(p_2E). \end{aligned}$$

Next, we give the following distributivity identities:

$$\begin{aligned} (E+F) \cdot_z G &= E \cdot_z G + F \cdot_z G, & p(E+F) &= pE + pF, \\ f(\cdots, E+F, \cdots) &= f(\cdots, E, \cdots) + f(\cdots, F, \cdots), & (p_1+p_2)E &= p_1E + p_2E. \end{aligned}$$

Moreover, we add the following identities involving $\mathbb{0}$:

$$f(\cdots, \mathbb{0}, \cdots) = \mathbb{0}, \qquad \mathbb{0} \cdot_z E = \mathbb{0}.$$

The semantics of pRTE is defined inductively on the structure of the expression: Let $t \in \mathrm{T}_\Sigma(V)$ we let $\|\mathbb{0}\|(t) = 0$, $\|z\|(t) = \mathbb{1}_{\{z\}}(t)$ for $z \in \Sigma_0 \cup V$, and

$$\|f(E_1,\dots,E_n)\|(t) = \begin{cases} \prod_{i=1}^n \|E_i\|(t_i) & \text{if } t = f(t_1,\dots,t_n) \text{ for } f \in \Sigma \cup V \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} \|E+F\|(t) &= \|E\|(t) + \|F\|(t) & \|pE\|(t) &= p\|E\|(t) \\ \|E \cdot_z f\|(t) &= (\|E\| \cdot_z \|F\|)(t) & \|E^{\infty z}\|(t) &= \|E\|^{\infty z}(t). \end{aligned}$$

Remark, that the syntax of pRTE is decidable. First check whether the input string is well-formed, then maximally expand it using the expanding distributivity rules (e.g. $(E+F)\cdot_z G \to E\cdot_z G + F\cdot_z G$). Finally, check the resulting string using the context-sensitive grammar arising from all rules except the expanding ones.

▶ **Example 9.** Let $\Sigma = \Sigma_2 \cup \Sigma_0$ with $\Sigma_2 = \{\mathsf{f}\}$ and $\Sigma_0 = \{\mathsf{a}, \mathsf{b}\}$. Furthermore, let y and z be variables. Consider the expression

$$E = \big(1/2\ \mathsf{f}(\mathsf{y},\mathsf{z}) + 1/2\ \mathsf{f}(\mathsf{z},\mathsf{y}) + \mathsf{a}\big)^{\infty_\mathsf{y}} \cdot_\mathsf{z} \big(\mathsf{f}(\mathsf{z},\mathsf{z}) + \mathsf{a} + \mathsf{b}\big)^{\infty_\mathsf{z}}.$$

Let the first factor be denoted by E_1 and the second factor by E_2. Then E_1 stochastically chooses a branch whose leaf nodes are labelled by a and z, as every variable y must be eventually substituted by a for the iteration to stop. Thus, E_1 assigns the probability $(1/2)^n$ to trees of the form $f_1(f_2(\cdots f_n(\mathsf{a})\cdots))$, where $f_i(t) = \mathsf{f}(t,\mathsf{z})$ or $f_i(t) = \mathsf{f}(\mathsf{z},t)$, and 0 to every tree not of this form. The expression E_2 assigns probability 1 to every tree. Thus, we obtain $\|E\|(t) = \sum_{x\in \mathrm{dom}_a(t)} (1/2)^{|x|}$.

3.3 Expressive equivalence to automata

The first part of this section will give an inductive construction to translate probabilistic regular tree expressions to top-down probabilistic tree automata. Afterwards, we show the converse direction. As a first step, we transfer the concept of substitution summability to automata.

▶ **Definition 10.** A probabilistic tree automaton $A = (Q, \delta, \mu, F)$ is called *substitution summable* if the $|V|+1$ sets F_{Σ_0}, $(F_{\{z\}})_{z\in V}$ are pairwise disjoint and every state in F_V is a sink state, where $F_W = \{\, q \in Q \mid (q,a) \in F \text{ for some } a \in W \,\}$ for $W \subseteq \Sigma_0 \cup V$.

▶ **Lemma 11.** *Let A be a PTA. If A is substitution summable, so is $\|A\|$.*

The class of substitution summable tree series is closed under the operations $\cdot_z$ and $^{\infty_z}$. The same statement holds for the class of substitution summable PTA.

▶ **Lemma 12.** *Let A_1, A_2 be PTA and A_1 be substitution summable. Then $\|A_1\| \cdot_z \|A_2\|$ can be recognized by a probabilistic tree automaton B. If A_2 is substitution summable, so is B.*

Proof. The construction is based on the weighted case, for details see [7]. The automaton B is the disjoint union of A_1, but with the states in $(F_1)_z$ removed, and A_2. The initial distribution of A_1 is used. Every transition into a state in $(F_1)_z$ is redirected to A_2 according to the initial distribution of A_2. Thus, the automaton B simulates runs of A_1 followed by runs of A_2. As A_1 is substitution summable, whenever a simulated run of A_1 could enter a state in F_z the only possible choice is to continue the run in A_2. The substitution summability of A_2 directly carries over to B. ◀

▶ **Lemma 13.** *Let A be a substitution summable PTA. Then $\|A\|^{\infty_z}$ is recognizable by a substitution summable PTA.*

Proof. Let $A = (Q, \delta, \mu, F)$ and $F_z = \{\, q \in Q \mid (q,z) \in F \,\}$. We may assume $\mu(F_z) < 1$. Let $\eta = \frac{1}{1-\mu(F_z)}$. We define the automaton $A' = (Q', \delta', \mu', F')$ by $Q' = Q \setminus F_z$, $\mu'(q) = \eta\mu(q)$, $F' = F \setminus F_z$, and

$$\delta'(q,f)(q_1,\ldots,q_n) = \sum_{r_1,\ldots,r_n\in Q} \delta(q,f)(r_1,\ldots,r_n) \prod_{i=1}^{n} \kappa(r_i,q_i),$$

where $\kappa(r,q) = \mathbb{1}_{\{q\}}(r) + \mathbb{1}_{F_z}(r)\eta\mu(q)$. The automaton A' simulates A until it can enter a state in F_z. As A is substitution summable the only possibility for A in such a state is to accept a leaf labelled with z. Instead, A' resets the simulated run of A using the initial distribution, thus starting a new iteration. The additional factor η is inserted to model arbitrary many substitutions of z by itself, each of them having the probability of $\mu(F_z)$. ◀

▶ **Lemma 14.** *Let E be a probabilistic regular tree expression. There is a substitution summable probabilistic tree automaton A such that $\|E\| = \|A\|$.*

Proof. We show that the set of expressions whose semantics is recognizable by an automaton satisfies all closure properties of pRTE and hence contains all expressions.

Clearly $\mathbb{0}$ and $\mathbb{1}_{\{z\}}$, for $z \in V$, are recognizable. Next, we consider the expression $E = \sum_{f\in\Sigma'} f(E_1^f, \dots, E_{ar(f)}^f)$ for some $\Sigma' \subseteq \Sigma$ and expressions E_i^f with $f \in \Sigma'$ and $1 \leq i \leq ar(f)$. Assume there are automata A_i^f with $\|A_i^f\| = \|E_i^f\|$ for each f and i. An automaton A recognizing E is constructed by taking the disjoint union of the automata A_f^i together with a new initial state q_0. If A reads a symbol $g \in \Sigma'$ with $ar(g) > 0$ in q_0, it simulates each automaton A_i^g at node i for $i = 1, \dots, ar(g)$. Furthermore, every symbol $a \in \Sigma' \cap \Sigma_0$ is accepted in q_0.

For $E = pE_1 + (1-p)E_2$ consider automata A_1 and A_2 recognizing $\|E_1\|$ and $\|E_2\|$, respectively. The automaton A is the disjoint union of A_1 and A_2, but with the initial distribution $p\mu_1 + (1-p)\mu_2$.

By Lemmas 12 and 13 we have the closure under tree concatenation and infinity iteration. Finally, note that the associativity, commutativity, and distributivity rules do not change the semantics of the expression and hence no automata construction is necessary. ◀

▶ **Example 15.** We consider the expression E from Example 9. Using the constructions described above we obtain a PTA recognizing $\|E\|$. The steps are shown in Figure 1:

(a) Probabilistic automata recognizing the constant series equal to 1

(b) The automata obtained for the expression $1/2\ (\mathsf{f}(\mathsf{y},\mathsf{z}) + \mathsf{a}) + 1/2\ (\mathsf{f}(\mathsf{z},\mathsf{y}) + \mathsf{a})$ using the constructions from Lemma 14.

(c) Lemma 13 is applied to the automaton from (b).

(d) Lemma 12 is applied to the automata from (c) and (a). This automaton recognizes $\|E\|$.

▶ **Lemma 16.** *Let A be a tree automaton. There is an expression E with $\|E\| = \|A\|$.*

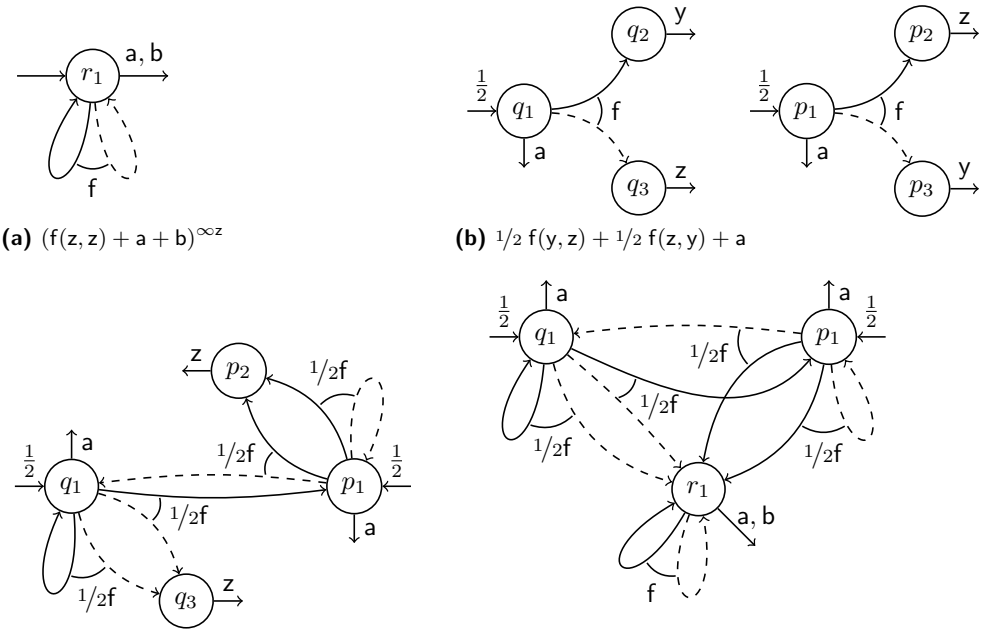
Proof. Let $A = (Q, \delta, \mu, F)$ and assume $Q = \{1, \dots, n\}$. Let $t \in \mathrm{T}_\Sigma(Q)$ and $i, k \in Q$. Define the set $R_i^k(t)$ to contain all runs $\rho\colon \mathrm{dom}(t) \to Q$ such that the following conditions hold:

1. $\rho(\varepsilon) = i$
2. $\rho(x) \leq k$ for all $x \in \mathrm{dom}_\Sigma(t) \setminus \{\varepsilon\}$
3. $\rho(x) = t(x)$ for all $x \in \mathrm{dom}_Q(t)$
4. $(\rho(x), t(x)) \in F$ for all $x \in \mathrm{dom}_{\Sigma_0}(t)$.

For a run ρ let $\mathrm{prob}(\rho) = \prod_{x\in\mathrm{inner}(t)} \delta(\rho(x), t(x))(\rho(x1), \dots, \rho(xn_x))$. We inductively construct expressions E_i^k over $\mathrm{T}_\Sigma(Q)$ such that

$$\|E_i^k\|(t) = \begin{cases} 0 & \text{if } t(\varepsilon) \in Q \text{ or } t(x) \leq k \text{ for a } x \in \mathrm{dom}_Q(t) \\ \sum_{\rho\in R_i^k(t)} \mathrm{prob}(\rho) & \text{otherwise.} \end{cases} \tag{1}$$

Intuitively, the index k is the largest state number that has already been handled in the construction. Trees must not contain handled states.



(a) $(\mathsf{f}(\mathsf{z},\mathsf{z}) + \mathsf{a} + \mathsf{b})^{\infty\mathsf{z}}$ **(b)** $1/2\ \mathsf{f}(\mathsf{y},\mathsf{z}) + 1/2\ \mathsf{f}(\mathsf{z},\mathsf{y}) + \mathsf{a}$

(c) $(1/2\ \mathsf{f}(\mathsf{y},\mathsf{z}) + 1/2\ \mathsf{f}(\mathsf{z},\mathsf{y}) + \mathsf{a})^{\infty\mathsf{y}}$ **(d)** $(1/2\ \mathsf{f}(\mathsf{y},\mathsf{z}) + 1/2\ \mathsf{f}(\mathsf{z},\mathsf{y}) + \mathsf{a})^{\infty\mathsf{y}} \cdot_{\mathsf{z}} (\mathsf{f}(\mathsf{z},\mathsf{z}) + \mathsf{a} + \mathsf{b})^{\infty\mathsf{z}}$

A transition $\eta = \delta(q, f)(q_1, q_2)$ is drawn as two arrows connected by an arc near the source state q. The arc is labelled by the probability η the letter f. The solid arrow leads to the first state q_1 and the dashed arrow to the second state q_2.

Figure 1 Inductive construction of a tree automaton.

For E_i^0 only trees of height at most 1 can have non-zero values. Thus, E_i^0 can be given directly for every $i \in Q$:

$$\begin{aligned} E_i^0 &= \sum_{\substack{c\colon \Sigma_{>0} \to Q^* \\ |c(f)| = ar(f)}} \left(\prod_{f \in \Sigma_{>0}} \delta(i, f)(c(f)) \right) \left(\sum_{f \in \Sigma_{>0}} f(c(f)) + \sum_{\substack{a \in \Sigma_0 \\ (i,a) \in F}} a \right) \\ &= \sum_{\substack{a \in \Sigma_0 \\ (i,a) \in F}} a + \sum_{f \in \Sigma_{>0}} \sum_{q_1, \ldots, q_{ar(f)} \in Q} \delta(i, f)(q_1, \ldots, q_{ar(f)})\ f(q_1, \ldots, q_{ar(f)}), \end{aligned}$$

where the first line can be directly constructed using the syntax from Definition 8 and the second line is obtained using distributivity and commutativity.

Now, assume the expressions E_i^k have already been constructed. Explicit calculation shows that the expression E_i^{k+1} defined by $E_i^{k+1} = E_i^k \cdot_{k+1} \left(E_{k+1}^k\right)^{\infty(k+1)}$ actually satisfies (1). We obtain the desired expression $E = \sum_{q \in Q} \mu(q) E_q^n$. ◀

Altogether, we have proven the following theorem:

▶ **Theorem 17.** *Let $S\colon \mathrm{T}_\Sigma \to [0,1]$ be a probabilistic tree series. The following statements are equivalent:*

1. *$S = \|A\|$ for some probabilistic tree automaton A,*
2. *$S = \|E\|$ for some probabilistic regular tree expression E.*

4 Probabilistic MSO Logic on Trees

In the first part of this section we give the formal definition of probabilistic MSO logic, basic properties, and an example. To prove the equivalence to probabilistic automata in the third part, we beforehand introduce bottom-up probabilistic tree automata in the second part and prove a Nivat-style theorem.

4.1 Syntax and Semantics of Probabilistic MSO logic

For the rest of this section fix a countable set of first order variables $\mathcal{V}_1$ and a countable set of second order variables $\mathcal{V}_2$. Let $\mathcal{V} = \mathcal{V}_1 \dot\cup \mathcal{V}_2$. Let $t \in \mathrm{T}_\Sigma$ be a tree. A *t-assignment* is a mapping $\sigma\colon \mathcal{V} \to \mathrm{dom}(t) \cup 2^{\mathrm{dom}(t)}$ such that $\sigma(\mathcal{V}_1) \subseteq \mathrm{dom}(t)$ and $\sigma(\mathcal{V}_2) \subseteq 2^{\mathrm{dom}(t)}$. We denote by $\sigma[x \mapsto a]$ the updated assignment which maps x to a and agrees with σ everywhere else.

▶ **Definition 18.** The set of all probabilistic MSO formulas φ is given in BNF by

$$\psi ::= \mathrm{label}_f(x) \mid \mathrm{edge}_i(x,y) \mid x \in X \mid \neg\psi \mid \psi \wedge \psi \mid \forall x.\psi \mid \forall X.\psi$$
$$\varphi ::= \psi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbb{E}_p X.\varphi\,,$$

where $f \in \Sigma$, $i \in \mathbb{N}$, $x \in \mathcal{V}_1$, $X \in \mathcal{V}_2$, and $p \in [0,1]$. The formulas generated by ψ are called Boolean formulas. The set $\mathrm{free}(\varphi)$ is defined as usual for Boolean MSO, conjunction, and negation. Additionally, we define $\mathrm{free}(\mathbb{E}_p X.\varphi) := \mathrm{free}(\varphi) \setminus \{X\}$.

The semantics of a formula φ is a function $[\![\varphi]\!]$ which maps a pair (t,σ), where t is a tree and σ is a t-assignment, to a probability value. The inductive definition is as follows:

$$[\![\psi]\!](t,\sigma) = \begin{cases} 1 & \text{if } (t,\sigma) \models \psi \\ 0 & \text{otherwise} \end{cases} \qquad [\![\varphi_1 \wedge \varphi_2]\!](t,\sigma) = [\![\varphi_1]\!](t,\sigma) \cdot [\![\varphi_2]\!](t,\sigma)$$
$$[\![\neg\varphi]\!](t,\sigma) = 1 - [\![\varphi]\!](t,\sigma)$$
$$[\![\mathbb{E}_p X.\varphi]\!](t,\sigma) = \sum_{M \subseteq \mathrm{dom}(t)} [\![\varphi]\!](t, \sigma[X \mapsto M]) \cdot p^{|M|}(1-p)^{|\mathrm{dom}(t)\setminus M|},$$

where $(t,\sigma) \models \psi$ is the usual satisfaction relation for classical MSO logic.

The semantics of the conjunction and negation are motivated from probability theory by the probabilities of the intersection of independent events and the complementary event, respectively. The semantics of $\mathbb{E}_p X.\varphi$ is as follows: We choose a set $M \subseteq \mathrm{dom}(t)$ using a sequence of independent Bernoulli experiments, i.e, for every tree position an unfair coin is tossed to decide whether to include this position in the set or not. For every such set the probability whether φ holds is computed. Finally, the expected value of these probabilities is calculated.

Disjunction can be defined as usual: $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$. The semantics is then given by $[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] + [\![\varphi_2]\!] - [\![\varphi_1]\!][\![\varphi_2]\!]$. This resembles the well-known identity for the probability of the union of independent events.

We write $\varphi_1 \equiv \varphi_2$ if $[\![\varphi_1]\!] = [\![\varphi_2]\!]$. The following identities are valid:

$$\psi \wedge (\varphi_1 \vee \varphi_2) \equiv (\psi \wedge \varphi_1) \vee (\psi \wedge \varphi_2) \qquad \mathbb{E}_p X.\neg\varphi \equiv \neg\mathbb{E}_p X.\varphi$$
$$\mathbb{E}_p X.\mathbb{E}_q Y.\varphi \equiv \mathbb{E}_q X.\mathbb{E}_p Y.\varphi \qquad \mathbb{E}_p X.\varphi \equiv \varphi \text{ if } X \notin \mathrm{free}(\varphi),$$
$$\mathbb{E}_p X.(\varphi_1 \wedge \varphi_2) \equiv (\mathbb{E}_p X.\varphi_1) \wedge \varphi_2 \text{ if } X \notin \mathrm{free}(\varphi_2)$$

where φ, φ_1, φ_2 are arbitrary probabilistic MSO formulas and ψ is Boolean.

Using these identities, the following statement can be shown:

▶ **Lemma 19.** *Let φ be a probabilistic MSO formula. There is a Boolean MSO formula ψ, second order variables $X_1, \dots, X_n$, and probability values $p_1, \dots, p_n$ such that $\varphi \equiv \mathbb{E}_{p_1} X_1 \cdots \mathbb{E}_{p_n} X_n.\psi$.*

▶ **Example 20.** Let $\Sigma = \Sigma_2 \cup \Sigma_0$ with $\Sigma_2 = \{\mathsf{g}\}$ and $\Sigma_0 = \{\mathsf{a}, \mathsf{b}\}$. We consider a search for a leaf labelled by a. This search works by iterating the leafs of the tree left to right. At every visited leaf the search stops with probability p. It is successful if the label of the node, where the search stopped, is a. This process can be modelled by the following formula:

$$\varphi = \mathbb{E}_p X.\exists x.x \in X \wedge \mathrm{label}_{\mathsf{a}}(x) \wedge \mathit{leaf}(x) \wedge \big(\forall y.(y \in X \wedge \mathit{leaf}(y)) \implies x \sqsubseteq_{\mathrm{DF}} y\big),$$

where $\sqsubseteq_{\mathrm{DF}}$ is the depth-first traversal order and *leaf* the predicate whether a node is a leaf, both are MSO definable. The Boolean part of φ expresses that the minimal leaf position in X is labelled by a. For the semantics we obtain

$$[\![\varphi]\!](t) = \sum_{x \in \mathrm{leaf}_{\mathsf{a}}(t)} p \cdot (1-p)^{n_x} \text{ where } n_x = |\{y \in \mathrm{leaf}(t) \mid y \sqsubseteq_{\mathrm{DF}} x,\, y \neq x\}|.$$

The tree series $[\![\varphi]\!]$ from Example 20 is not recognizable by a top-down probabilistic tree automaton. Intuitively, when a top-down automaton reaches a leaf node, there is no information available whether there are a-labelled leafs to the left of this node. Therefore, we will define a more expressive probabilistic automata model in the next section.

▶ Remark. The syntax of probabilistic MSO was chosen to be minimal. In fact, some constructs known from weighted MSO logics can be expressed in probabilistic MSO as syntactic macros, i.e., they can be transformed to the syntax of Definition 18.

Multiplication by constants: Consider two PMSO formulas φ and ψ and a probability value p. The formula $\mathbb{E}_p X.(\exists x.\mathit{root}(x) \wedge (x \in X \implies \varphi) \wedge (x \notin X \implies \psi)$, where x and X are new variable symbols and $\mathit{root}(x)$ is a MSO formula, which checks if x is the root position, has as semantics $p[\![\varphi]\!] + (1-p)[\![\psi]\!]$.

Generalised universal first-order quantification: Weighted logics allows universal first order quantification to be applied to arbitrary formulas. We can also introduce such an operator to probabilistic logic. Let φ be a probabilistic MSO formula. The semantics of $\forall x.\varphi$ is given by $[\![\forall x.\varphi]\!](t,\sigma) = \prod_{a \in \mathrm{dom}(t)} [\![\varphi]\!](t, \sigma[x \mapsto a])$. As in the weighted case, $\forall x.$ does not preserve recognizability when arbitrary formulas φ are allowed. Thus, φ is restricted to so-called step formulas. A step formula is build from Boolean MSO formulas and probability constants using only the Boolean operations. Thus, every step formula φ can be written as $\varphi \equiv \bigwedge_{i=1}^{n} (\psi_i \implies p_i)$. We introduce new second order variables $X_1, \dots, X_n$ and define a formula φ' by

$$\varphi' \coloneqq \mathbb{E}_{p_1} X_1. \cdots \mathbb{E}_{p_n} X_n. \forall x. \bigwedge_{i=1}^{n} (\psi_i \implies x \in X_i).$$

It can be shown that $[\![\varphi']\!] = [\![\forall x.\varphi]\!]$ holds.

4.2 A Nivat Theorem

We introduce bottom-up deterministic tree automata, which are a generalisation of bottom-up deterministic tree automata to the probabilistic setting.

▶ **Definition 21.** A *bottom-up probabilistic tree-automaton* is a triple $A = (Q, \delta, F)$ where
1. Q is a finite, non-empty set – the set of states,
2. $\delta = \bigcup_{n\geq 0} \delta_n$ where $\delta_n \colon \Sigma \times Q^n \to \Delta(Q)$ – the transition probability function,
3. $F \subseteq Q$ – the set of final states.

The semantics is given by $\|A\|(t) = \sum_{q\in F} \delta_q(t)$, where

$$\delta_q\big(f(t_1,\dots,t_n)\big) = \sum_{q_1,\dots,q_n\in Q} \delta(f,q_1,\dots,q_n)(q) \prod_{i=1}^{n} \delta_{q_i}(t_i).$$

Although bottom-up PTA are a natural generalisation of deterministic bottom-up tree automata to the probabilistic setting, we only found one other reference to this model [12].

Before we state our Nivat theorem, we introduce some notation: Let Γ be another rank alphabet. A mapping $h \colon \Gamma \to \Sigma$ is called a *relabelling* if $ar_\Gamma(g) = ar_\Sigma(h(g))$ for all $g \in \Gamma$. A relabelling extends uniquely to a function $h \colon \mathrm{T}_\Gamma \to \mathrm{T}_\Sigma$ by $\mathrm{dom}(h(t)) = \mathrm{dom}(t)$ and $h(t)(x) = h(t(x))$ for all $x \in \mathrm{dom}(t)$. Given a finite set M, we can interpret $M \times \mathbb{N}_0$ as an (infinite) rank alphabet by defining $ar((m,k)) = k$. For a tree domain D, a finite set M and a distribution d on M, we define a probability measure $\Pr_d^D$ on $\{t \in \mathrm{T}_{M\times\mathbb{N}_0} \mid \mathrm{dom}(t) = D\}$ by $\Pr_d^D(\{t\}) = \prod_{x\in D} d(\pi_1(t(x)))$, where π_1 is the projection on the first component. Given a relabelling $g \colon \Sigma \to M \times \mathbb{N}_0$ we let $(\Pr_d^D \circ g)(X) = \Pr_d^D(g(X))$ for all $X \subseteq \{t \in \mathrm{T}_\Sigma \mid \mathrm{dom}(t) = D\}$. We write $\Pr_d$ instead of $\Pr_d^D$ if D is understood.

▶ **Theorem 22.** *Let $S \colon \mathrm{T}_\Sigma \to [0,1]$ be a probabilistic tree series.*
1. *S is the behaviour of a bottom-up probabilistic tree automaton if and only if there are*
   - **(a)** *a finite rank alphabet Γ and a finite set M,*
   - **(b)** *a distribution d on M,*
   - **(c)** *relabellings $h \colon \Gamma \to \Sigma$ and $g \colon \Gamma \to M \times \mathbb{N}_0$,*
   - **(d)** *a regular tree language $L \subseteq \mathrm{T}_\Gamma$,*

   such that for all $t \in \mathrm{T}_\Sigma$

$$\|A\|(t) = (\Pr_d \circ g)\big(h^{-1}(\{t\}) \cap L\big)\,. \tag{2}$$

2. *S is the behaviour of a top-down probabilistic tree automaton if and only if conditions a – d hold and additionally*
   - **(e)** *L is top-down deterministic recognizable,*
   - **(f)** *the mapping $\Gamma \to \Sigma \times M$ defined by $f \mapsto (h(f), g(f))$ is injective,*

   and (2) holds.

Proof. We only prove the bottom-up case, the top-down case in analogous. Additional care has to be taken, as $\Pr_d$ contains a probability distribution for every tree node, whereas a top-down PTA only contains one for the inner nodes.

Let $A = (Q, \delta, F)$ be a bottom-up PTA. Define the set M by $M = \prod_{n\geq 0} M_n$ where $M_n = Q^{\Sigma_n \times Q^n}$, i.e., M contains functions mapping tuples $(f, q_1, \dots, q_{ar(f)})$ to states. The probability distribution d is given by

$$d(m) = \prod_{f\in\Sigma} \prod_{q_1,\dots,q_{ar(f)}\in Q} \delta(f,q_1,\dots,q_{ar(f)})(m(f,q_1,\dots,q_{ar(f)})).$$

Let $\Gamma = \Sigma \times M$ with $ar(f,m) = ar(f)$, we set $h(f,m) = f$, and $g(f,m) = (m, ar(f))$. Let the tree language L contain all trees $t \in \mathrm{T}_\Gamma$ for which there is a run $\rho \colon \mathrm{dom}(t) \to Q$ with $\rho(x) = \pi_2(t(x))(\pi_1(t(x)), \rho(x\,1), \dots, \rho(x\ ar(t(x))))$ for all $x \in \mathrm{dom}(t)$, and $\rho(\varepsilon) \in F$. Then, L is regular. One can check that (2) holds using these definitions.

Conversely, consider the rank alphabet $\Gamma' = \Sigma \times M$ and the relabelling $\kappa\colon \Gamma \to \Gamma'$ with $\kappa(a) = (h(a), \pi_1(g(a)))$. Let $A' = (Q', \delta', F')$ be a deterministic bottom-up tree automaton with $\mathrm{L}(A') = \kappa(L)$. We define $A = (Q', \delta, F')$ with $\delta(f, q_1, \ldots, q_n) = \sum\{d(m) \mid \delta'((f,m), q_1, \ldots, q_n) = q\}$. Again, we obtain (2). ◀

Using Theorem 22 we immediately conclude the following corollary.

▶ **Corollary 23.** *Let A be a top-down probabilistic tree automaton. There is a bottom-up probabilistic tree automaton B with $\|B\| = \|A\|$.*

4.3 Equivalence to Tree Automata

▶ **Theorem 24.** *Let $S\colon \mathrm{T}_\Sigma \to [0,1]$. The following statements are equivalent:*

1. *$S = \|A\|$ for a probabilistic bottom-up tree automaton A,*
2. *$S = [\![\varphi]\!]$ for a probabilistic MSO sentence φ.*

Proof. Given a bottom-up PTA A, we apply Theorem 22 to obtain Γ, M, L, d, h and g as in the statement of the theorem such that (2) holds. We may assume $M = \{1, \ldots, m\}$ and $\Gamma = \{1, \ldots, \ell\}$. Choose probability values $p_1, \ldots, p_m$ such that $d(k) = p_k \prod_{i=1}^{k-1}(1-p_i)$. Using the classical Büchi theorem one constructs a Boolean MSO sentence ψ such that $\mathrm{L}(\psi) = L$. Let $X_1, \ldots, X_m, Y_1, \ldots, Y_\ell$ be new set variable symbols. Replace every occurrence of $\mathrm{label}_f(x)$ in ψ by $x \in Y_f$ resulting in a new formula ψ'. Let $\mathit{part}(Y_1, \ldots, Y_k)$ be a MSO formula expressing that $Y_1, \ldots, Y_k$ is a partition of the domain. We define φ as

$$\varphi = \mathbb{E}_{p_1} X_1 \cdots \mathbb{E}_{p_m} X_m \exists Y_1 \cdots \exists Y_\ell . \mathit{part}(Y_1, \ldots, Y_\ell) \wedge \psi'$$
$$\wedge \forall x . \bigwedge_{f \in \Gamma} x \in Y_f \implies \Big(\mathrm{label}_{h(f)}(x) \wedge x \in X_{g(f)} \wedge \bigwedge_{k=1}^{g(f)-1} x \notin X_k\Big).$$

Consider a tree $t \in \mathrm{T}_\Sigma$. The Y_i's encode a tree $\bar{t} \in \mathrm{T}_\Gamma$ with $\bar{t} \models \psi$, i.e., $\bar{t} \in L$. The second line of the formula states that $t = h(\bar{t})$ and chooses the X_i's such that the minimal k with $x \in X_k$ is $g(\bar{t}(x))$ for every $x \in \mathrm{dom}(t)$. Hence, fixing $\bar{t}$, the probabilities at every position x sum up to $p_{g(\bar{t}(x))} \prod_{i=1}^{g(\bar{t}(x))-1}(1-p_i) = d(g(\bar{t}(x)))$. Thus, we obtain $\Pr_d(\{g(\bar{t})\})$ for the whole tree. Considering arbitrary $\bar{t}$, we conclude that $[\![\varphi]\!]$ equals the right side of (2).

Conversely, let φ be a probabilistic MSO sentence. By Lemma 19, there is an equivalent sentence of the form $\mathbb{E}_{p_1} X_1 \cdots \mathbb{E}_{p_m} X_m . \psi$ where ψ is Boolean. Let $M = \{0,1\}^m$, $\Gamma = \Sigma \times M$, and $h\colon \Gamma \to \Sigma$ and $g\colon \Gamma \to M$ be the natural projections. Define $d \in \Delta(M)$ by $d(m) = (\prod_{i,\, m(i)=1} p_i)(\prod_{i,\, m(i)=0}(1-p_i))$. Again, by the classical Büchi theorem $L = \mathrm{L}(\psi) \subseteq \mathrm{T}_\Gamma$, where the additional components of $\Sigma \times \{0,1\}^m = \Gamma$ encode the values of the X_i's, is regular. One shows that (2) holds in this situation. Thus, by Theorem 22, there is a bottom-up PTA recognizing $[\![\varphi]\!]$. ◀

5 Conclusion and Future Research

We have introduced a probabilistic variant of regular tree expressions and proved that these expressions are expressively equivalent to top-down probabilistic tree automata. Next, we gave an extension of MSO logic to the probabilistic setting. It turned out that top-down PTA are too weak to recognize the semantics of all probabilistic MSO sentences. Thus, we introduced bottom-up probabilistic tree automata. We could show that the class of these automata is expressively equivalent to probabilistic MSO. In order to prove this result we also obtained a Nivat-style theorem for bottom-up PTA.

Future research might look into an extension of these results to unranked trees. There already exists MSO logic on unranked trees for the unweighted [15] as well as for the weighted case [10]. For regular tree expressions there already exist forest expressions [2] and one could extend unweighted ranked regular tree expressions to the unranked case. None of these concepts directly fit into the probabilistic setting. A different, interesting structure is infinite ranked trees. Probabilistic tree automata for infinite trees have been given in [5]. The extension of probabilistic regular tree expressions to infinite trees looks promising. For probabilistic MSO logic there does not seem to be a proper automata model, but one could still get the equivalence to the tree series defined by the Nivat decomposition from Theorem 22. A different notion of probabilistic regular expressions on trees has been given in [14]. These expressions use pebbles and are tree-walking. It has been shown [3] that in the unweighted case pebble tree-walking automata are strictly less expressive than regular tree languages. It remains to be seen if this inclusion also holds in the probabilistic case.

References

1 Parvaneh Babari and Manfred Droste. A Nivat theorem for weighted picture automata and weighted MSO logic. In *Proc. of LATA 2015*, volume 8977 of *LNCS*, pages 703–715. Springer, 2015.

2 Mikołaj Bojańczyk. Forest expressions. In *Computer Science Logic*, volume 4646 of *LNCS*, pages 146–160. Springer, 2007.

3 Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebble automata. In *ICALP 2006*, volume 4051 of *LNCS*, pages 157–168. Springer, 2006.

4 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. A probabilistic Kleene theorem. In *Proc. of ATVA 2012*, volume 7561 of *LNCS*, pages 400–415. Springer, 2012.

5 Arnaud Carayol, Axel Haddad, and Olivier Serre. Randomization in automata on infinite trees. *ACM Trans. Comput. Log.*, 15(3):24, 2014.

6 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2008. release November 18, 2008.

7 Manfred Droste, Christian Pech, and Heiko Vogler. A Kleene theorem for weighted tree automata. *Theory of Computing Systems*, 38(1):1–38, 2005.

8 Manfred Droste and Vitaly Perevoshchikov. A Nivat theorem for weighted timed automata and weighted relative distance logic. In *Proc. of ICALP 2014*, volume 8573 of *LNCS*, pages 171–182. Springer, 2014.

9 Manfred Droste and Heiko Vogler. Weighted tree automata and weighted logics. *Theoretical Computer Science*, 366(3):228 – 247, 2006. Automata and Formal Languages.

10 Manfred Droste and Heiko Vogler. Weighted logics for unranked tree automata. *Theory Comput. Syst.*, 48(1):23–47, 2011.

11 Clarence A. Ellis. Probabilistic tree automata. *Information and Control*, 19(5):401 – 416, 1971.

12 Olympia Louscou-Bozapalidou. Stochastic tree functions. *International Journal of Computer Mathematics*, 52(3-4):149–160, 1994.

13 M. Magidor and G. Moran. Probabilistic tree automata and context free languages. *Israel Journal of Mathematics*, 8(4):340–348, 1970.

14 Benjamin Monmege. *Specification and verification of quantitative properties: expressions, logics, and automata.* PhD thesis, ENS Cachan, 2013.

15 Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theor. Comput. Sci.*, 275(1-2):633–674, 2002.

16 Maurice Nivat. Transductions des langages de Chomsky. *Annales de l'institut Fourier*, 18(1):339–455, 1968.

17 Thomas Weidner. Probabilistic automata and probabilistic logic. In *Proc. of MFCS 2012*, volume 7464 of *LNCS*, pages 813–824. Springer, 2012.

18 Thomas Weidner. Probabilistic ω-regular expressions. In *Proc. of LATA 2014*, volume 8370 of *LNCS*, pages 588–600. Springer, 2014.

Rumors Across Radio, Wireless, and Telephone*

Jennifer Iglesias†[1], Rajmohan Rajaraman[2], R. Ravi[1], and Ravi Sundaram[2]

1 **Carnegie Mellon University**
5000 Forbes Avenue, Pittsburgh, PA, USA
{jiglesia, ravi}@andrew.cmu.edu

2 **Northeastern University**
360 Huntington Ave., Boston, MA, USA
{rraj, koods}@ccs.neu.edu

Abstract

We study the problem of computing a minimum time schedule to spread rumors in a given graph under several models: In the radio model, all neighbors of a transmitting node listen to the messages and are able to record it only when no other neighbor is transmitting; In the wireless model (also called the edge-star model), each transmitter is at a different frequency to which any neighbor can tune to, but only one neighboring transmission can be accessed in this way; In the telephone model, the set of transmitter-receiver pairs form a matching in the graph. The rumor spreading problems assume a message at one or several nodes of the graph that must reach a target node or set of nodes. The transmission proceeds in synchronous rounds under the rules of the corresponding model. The goal is to compute a schedule that completes in the minimum number of rounds.

We present a comprehensive study of approximation algorithms for these problems, and show several reductions from the harder to the easier models for special demands. We show a new hardness of approximation of $\Omega(n^{\frac{1}{2}-\epsilon})$ for the minimum radio gossip time by a connection to maximum induced matchings. We give the first sublinear approximation algorithms for the most general case of the problem under the wireless model; we also consider various special cases such as instances with symmetric demands and give better approximation algorithms. Our work exposes the relationships across the models and opens up several new avenues for further study.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Broadcast, Gossip, Approximation algorithms, Graph algorithms, Hardness of Approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.517

1 Introduction

Problems modeling rumor spread are central to the design of coordination networks that seek to keep demand pairs of vertices in contact over time. The prototypical example is the *broadcast* problem where a message in a root node must be sent to all the other nodes via connections represented by an undirected graph. We assume that communication proceeds in

* Research supported in part by ONR grant N000141211001 and NSF grants CCF-1422715 and CCF-1527032.

† This material is based n part upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 2013170941, and by Microsoft Research Graduate Women's Fellowship.

synchronized rounds. When more than one message is being disseminated, we assume that in each round each node can transmit an unlimited number of messages in one communication. A subset generalization of broadcast is called the *Multicast* problem: a subset of nodes is specified as terminals and the goal is to spread the rumor from the root only to this subset, using other non-terminal nodes if needed in the process. An all-to-all generalization of the broadcast problem is termed *gossip*: every node has its own piece of information that must be communicated to all nodes, and the goal is to have all the information spread to all the nodes in the minimum number of rounds. Gossip and broadcast are special cases of a more general demand model that we may call *multicommodity multicast*: in this most general version, we are given a set of source-sink pairs so that each source has a rumor that must be sent to the corresponding sink. Recall that messages from many sources can all be aggregated and exchanged in one round between any pair that can communicate, and the goal is to minimize the number of rounds. In this paper, we will study a specialization of the multicommodity demand model called the *symmetric multicommodity* where for every source-sink pair, we also have the symmetric requirment that the sink wants to send its rumor to the source; thus, the demand pairs are unordered in this case. The more general version will be called the *asymmetric multicommodity* demand model to distinguish it from the symmetric demands case.

1.1 Models: Telephone, Radio, and Edge-Star, a New Model from Wireless

Different communication models result in different constraints on the set of edges on which messages can be transmitted in a single round. The two most widely studied models are the telephone and radio models: In the **telephone** model, in each round, a node can communicate with at most one other node, thus the edges on which communication occurs is a matching; In the **radio** model, a set of transmitters broadcast the message out but only their neighbors who are adjacent to exactly only one transmitter can successfully receive the message (while interference prevents other neighbors from receiving the message): the set of edges through which the messages are sent in any round in this model is a set of stars centered at the transmitters, where each leaf of each star has that star's center as its unique neighbor among all the star centers.

In this paper, we expand the study of rumor spreading problems by introducing a new model based on wireless communications between nodes, which we call the **edge-star** model. We assume that during each round of wireless communication, each transmitter can choose its own channel or frequency distinct from that of all other transmitters. The input undirected graph represents pairs of nodes that are within wireless range of each other. Receiving nodes that are in the vicinity of many different transmitting nodes can choose to tune into the frequency of one of them. In this way, the set of edges in which communication happens in every round is a set of stars which are defined by a subset of edges of the input graph. Note that unlike the radio model, there is no requirement that a receiver be adjacent to exactly one transmitter. This model more closely models wireless networks, where a machine may be able to see many wireless networks, but only interacts with one of these networks at a time.

1.2 Previous Work

The radio broadcast and gossip problems have been extensively studied (see the work reviewed in the survey [11]). The best-known scheme for radio broadcast is by Kowalski and Pelc [12] which completes in time $O(D+\log^2 n)$, where n is the number of nodes, and D is the diameter

of the graph and is a lower bound to get the message across the graph from any root. The $O(\log^2 n)$ term is also unavoidable as demonstrated by Alon et al. [1] in an example with constant diameter that takes $\Omega(\log^2 n)$ rounds for an optimal broadcast scheme to complete. Elkin and Korsartz [5] also show that this additive log-squared term is best possible unless $NP \subseteq DTIME(n^{\log\log n})$.

The best bound for radio gossip known so far, however, is $O(D + \Delta \log n)$ steps in an n-node graph with diameter D and maximum degree Δ [10]. The maximum degree is not a lower bound on the gossip time, and indeed no previous results are known about the approximability for radio gossip, which is mentioned as an open problem in [11].

In the telephone model, the first poly-logarithmic approximation for minimum broadcast time was achieved by Ravi [14] and the current best known approximation ratio is $O(\frac{\log n}{\log\log n})$ due to Elkin and Korsartz [6]. The best known lower bound on the approximation ratio for telephone broadcast is $3 - \epsilon$ [4].

In his study of the telephone broadcast time problem, Ravi [14] introduced the idea of finding low poise spanning trees to accomplish broadcast: the *poise* of a spanning tree of an undirected graph is the sum of its diameter and its maximum degree. In the course of deriving a poly-logarithmic approximation, Ravi also showed how a tree of poise P in an n-node graph can be used to complete broadcast starting from any node in $O(P \cdot \frac{\log n}{\log\log n})$ steps - we will use this observation later.

Nikzad and Ravi [13] studied the telephone multicommodity multicast problem, and gave the first sub-linear approximation algorithm with performance ratio $2^{O(\log\log k\sqrt{\log k})}$ for instances with k source-sink pairs.

Gandhi et al. [9] recently studied the Radio Aggregation Scheduling problem which is a gathering version of the rumor spreading problem in the radio model. The set of edges in which communication occurs in every round is a matching with the additional property that if the edges within receivers and within senders are ignored, the communicating edges form an induced matching. In this model they prove a tight $\Theta(n^{1-\epsilon})$-approximation for their radio aggregation scheduling. Our results were derived independently of their methods.

1.3 Our contributions

We give the first results on the approximability of gossip and multicommodity multicast problems in the radio model. We introduce the edge-star model based on wireless channels and give the first approximation results for minimum time rumor spreading by relating them to their analogs in the telephone model.

1. We show that it is NP-hard to approximate gossip in the radio model within a factor of $O(n^{1/2-\epsilon})$ in an n-node graph. This result is derived by isolating a gathering version of the broadcast problem in the radio model and relating it in a simple bipartite graph to induced matchings (Section 2).
2. We obtain an $O(\frac{\log n}{\log\log n})$ approximation algorithm for gossip in the edge-star model by reducing the problem to the broadcast problem in the telephone model (Section 3.1).
3. We consider the special case where the underlying graph is a tree, and show that the multicommodity multicast in the edge-star model reduces to the broadcast problem in the telephone model, thus proving an $O(\frac{\log n}{\log\log n})$ approximation (Section 3.2).
4. We show that the case of edge-star symmetric multicommodity multicast problem has the same optimal solution (up to poly-log factors) as telephone multicommodity multicast, yielding a $2^{O(\log\log n\sqrt{\log n})}$ approximation (Section 3.3).

Table 1 A summary of upper and lower bounds achieved in the different problems. We prove the results marked * in the table.

	Broadcast	Gossip	Multicommodity
Radio	$D + O(\log^2 n)$ [12]	$O(D + \Delta \log n)$ [10]	Unknown
		$\Omega(n^{1/2-\epsilon})$ hard*	$\Omega(n^{1/2-\epsilon})$ hard*
Edge-star	OPT$= D$	OPT$\cdot O(\frac{\log n}{\log\log n})$*	OPT$\cdot \tilde{O}(2^{\sqrt{\log n}})$*(symmetric)
			OPT$\cdot O(n^{\frac{2}{3}})$* (asymmetric)
Telephone	OPT$\cdot O(\frac{\log n}{\log\log n})$ [7]	OPT$\cdot O(\frac{\log n}{\log\log n})$ [7]	OPT$\cdot \tilde{O}(2^{\sqrt{\log n}})$ [13]

5. We give an $O(n^{\frac{2}{3}})$-approximation for the general (asymmetric) multicommodity multicast problem in the edge-star model (Section 3.4).

Table 1 contains a summary of our results in context.

2 Lower bound for gossip in the radio model

In this section, we show it is NP-hard to approximate gossip in the radio model within a factor of $O(n^{1/2-\epsilon})$. This also implies the same hardness result for multicommodity multicast under the radio model, because gossip is a special case of multicommodity multicast. In order to show these hardness results, we first consider the smallest set of induced matchings which cover the vertices of a bipartite graph.

▶ **Definition 2.1.** An *induced matching* is a matching of some vertices U in a graph G, such that $G[U]$ is a matching. (We use $G[U]$ to mean the graph G induced on the vertex set U.) In other words, in the graph G only the matching edges are present between the nodes in U.

A *covering set of induced matchings* (CSIM) is a set of induced matchings which cover all the vertices in the graph. The size of a covering set of induced matchings is defined to be the number of induced matchings.

First, we will show the hardness of finding a minimum CSIM by a reduction from coloring. Then we will use the hardness of minimum sized CSIM to prove the hardness results for radio gossip.

▶ **Theorem 2.2.** *It is NP-hard to approximate CSIM to within a $n^{1/2-\epsilon}$ factor for any constant $\epsilon > 0$.*

Proof. Given a coloring instance $G = (V, E)$, we first turn this into a bipartite graph, where we want to find a CSIM. For each $v \in V$ we make $n+1$ copies of v in each side of the partition; $v_1^L, v_2^L, \ldots v_{n+1}^L$ for L and $v_1^R, v_2^R, \ldots v_{n+1}^R$ for R. We use the edges $E_v = \{(v_i^L, v_i^R) | v \in V, i \in [n+1]\}$, called the straight edges and $E_e = \{(u_i^L, v_j^R) | uv \in E, i, j \in [n+1]\}$, called the cross edges. Now $G' = (L, R, E_v \cup E_e)$ is the bipartite graph for which we want to find a CSIM. Figure 1 shows an example construction.

Let χ be the number of colors in an optimal coloring in G. Let λ be the number of sets in a minimal CSIM in G'.

We now show that $\lambda \leq \chi \leq n$. Let C_i be a set of vertices of color i in the coloring on G. If we take the edges $\{(v_j^L, v_j^R) | v \in C_i, j \in [n+1]\}$, they are an induced matching. Each vertex has one straight edge in G', and if a vertex is used in the matching, then its straight edge is used. So, we only need to show that no cross edges go between vertices in this matching. If

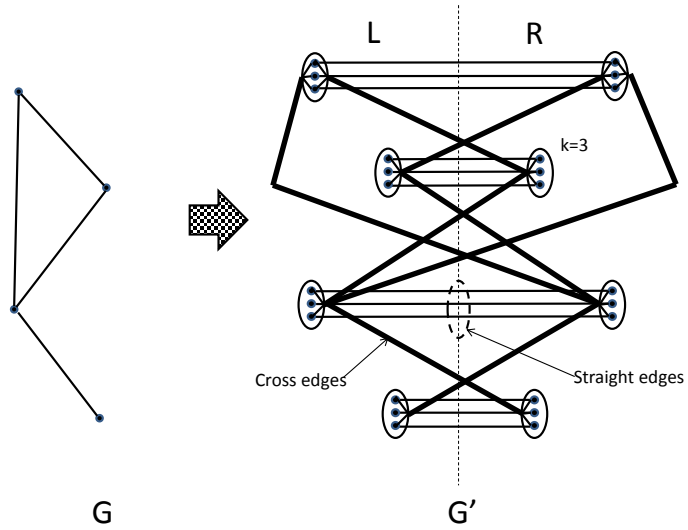


Figure 1 Here is an example of the construction of G' from G. The thick edges represent complete bipartite subgraphs.

a cross edge (u_j^L, v_k^R) did exist, then $(u, v) \in E$ but then u, v couldn't be the same color. So, for each color we have defined an induced matching. These induced matchings cover all the nodes since every node receives some color in the coloring on G.

Now we will show that $\chi \leq \lambda$ or $n + 1 \leq \lambda$. Let $S_1, S_2, \ldots S_\lambda$ be the induced matchings covering G'. Assume that there is some $v \in V$ that has all of its corresponding vertices in G' matched via cross edges. Then we can only have at most one cross edge per induced matching adjacent to the v_i^L's. If an induced matching has (v_i^L, u_ℓ^R) and $(v_j^L, w_{\ell'}^R)$ then this is not an induced matching since (v_j^L, u_ℓ^R) is an edge. Therefore in this case to match all the v_i^L in some induced matching, we will need at least $n + 1$ induced matchings. Now consider each $v \in G$ has one of its straight edges used in some induced matching. Let S_j be the first induced matching containing a straight edge adjacent to some v_i^L. In S_j, because some v_i^L is matched via its straight edge, then no v_ℓ^L is matched via a cross edge. Now in G color v with the jth color. This is a valid coloring. If some (v_i^L, v_i^R) and (u_ℓ^L, u_ℓ^R) were both in the same induced matching, then there can't be the edge (u, v) in the original graph G.

Combining the above two parts we get that $\chi = \lambda$.

We begin with a graph G such that it is NP-hard to distinguish if there is a coloring of size $|V(G)|^\epsilon$ from if the coloring requires at least $|V(G)|^{1-\epsilon}$ colors [8]. Therefore, in the graph G' we created, it is NP-hard to distinguish if there is a set of induced matchings that cover the vertices of size n^ϵ or n. We have $O(n^2)$ vertices in G' though. So, in a graph with n vertices it is NP-hard to approximate the number of induced matchings needed to cover the vertices within a factor of $O(n^{1/2-\epsilon})$. ◀

Now that we have developed the hardness result for CSIM, we will use the graph we created for CSIM, to create instances of radio gossip.

▶ **Corollary 2.3.** *It is NP-hard to approximate radio gossip to within a $n^{1/2-\epsilon}$ factor for any constant $\epsilon > 0$.*

Proof. We convert the induced matching instance to a gossip problem in a similar fashion

to above. We can consider that we have the bipartite graph G' and we build a complete binary tree with its leaves being the nodes v_i^L. The terminal nodes in the gossip problem are set to be all the nodes. To communicate the message to all other nodes, each node v_i^R must at some point be the only node trying to talk to some node on the other side of the bipartition. In other words, we need to have induced matchings at each point in order for the v_i^R to propagate their messages to some other node without interference. Therefore, we need at least as many induced matchings as it takes to cover the graph to complete the gossip. Call this number C; we can now achieve gossip in time $2C + 3\log n$ as follows. We do this by using the induced matchings so that each vertex v_i^R communicates its message to someone on the other side of the partition. Next we propagate the message up the binary tree to the root node. This takes time at most $2\log n$ since at each node of the path in the binary tree, a message can be delayed only for two steps, and the path length is logarithmic. Then we broadcast the message down the tree. This takes time $\log n$ since we can use the edge-star model to just broadcast all the gathered messages from the root along the down-stars in one time step per level. Lastly, we need to communicate the message back to the v_i^R, which takes time C. We know that radio gossip takes time at least C and can be done in time $2C + 3\log n$ on this graph.

Therefore, it is NP-hard to approximate radio gossip better than a factor of $O(n^{1/2-\epsilon})$ otherwise, we could approximate the CSIM within the same factor. ◀

3 The Edge-Star Model

In this section, we consider the edge-star model which generalizes the telephone model. We focus on three specific classes of problems; gossip, symmetric multicommodity multicast, and asymmetric multicommodity. In the *symmetric multicommodity* problem, we are given a set of demand pairs, and if (s_i, t_i) is a demand, then (t_i, s_i) is also a demand. In the asymmetric multicommodity case, there are no restrictions on which demand pairs are present.

In Section 3.1, we first obtain an $O(\frac{\log n}{\log\log n})$ approximation algorithm for gossip in the edge-star model by reducing the problem to the broadcast problem in the telephone model. Next, in Section 3.2, we consider the special case where the underlying graph is a tree. In this special case, then we show that the multicommodity multicast in the edge-star model reduces to the broadcast problem in the telephone model, yielding an $O(\frac{\log n}{\log\log n})$ approximation. In Section 3.3, we show that the case of edge-star symmetric multicommodity multicast problem has the same optimal solution (up to poly-log factors) as telephone multicommodity multicast, yielding an $\tilde{O}(2^{\sqrt{\log n}})$ approximation. Lastly, in Section 3.4, we give an $O(n^{\frac{2}{3}})$-approximation for the general (asymmetric) multicommodity multicast problem in the edge-star model.

3.1 Gossip

Here we show an $O(\frac{\log n}{\log\log n})$ approximation for edge-star gossip. First, we show that a solution to the gossip problem in the edge-star model gives a solution to the broadcast problem in the telephone model of the same length. Next, we show that using a solution for the broadcast problem in telephone we can get a solution of twice the length to the gossip problem in the edge-star model. This show that their optimal solutions differ in cost by a factor of at most two.

▶ **Lemma 3.1.** *The optimal broadcast time in the telephone model is no more than the optimal gossip time in the edge-star model.*

Proof. Let $\mathcal{S}$ denote an optimal schedule for gossip in the edge-star model that completes in T rounds. Let r denote the root node for the broadcast problem in the telephone model. Fix a node v. Let P_v denote a path taken by the message from v to arrive at r in the schedule S. Let E_t denote the set of all directed edges in $\cup_v P_v$ that are activated in round t in S. By definition of the edge-star model, if (u_1, v_1) and (u_2, v_2) are in E_t, then $v_1 \neq v_2$. Furthermore, by our choice of the paths, we obtain that (i) for any distinct (u_1, v_1) and (u_2, v_2) in E_t, $u_1 \neq u_2$; and (ii) the edges of P_v appear in order of increasing time in the collection of E_ts.

We now argue that a reverse schedule in which the activated sets are given by $E'_t = \text{REV}(E_{T-t})$ forms a broadcast schedule from the root, where $\text{REV}(X)$ equals $\{(v, u) : (u, v) \in X\}$ for any set X of directed edges. In any round t, for any distinct (u_1, v_1) and (u_2, v_2) in E_t, we have $u_1 \neq u_2$ and $v_1 \neq v_2$; therefore, $\text{REV}(E_t)$ is a matching. Since the edges of P_v appear in order of increasing time in the collection of E_ts, the edges of the $\text{REV}(P_t)$ appear in order of increasing time in the collection of E'_ts. Consequently, the message from the root is delivered to each node in T rounds. ◀

▶ **Lemma 3.2.** *The optimal gossip time in the edge-star model is no more than twice the optimal broadcast time in the telephone model.*

Proof. The proof mirrors the proof of Lemma 3.1. Let $\mathcal{S}$ denote an optimal schedule for broadcast from root r in the telephone model that completes in T rounds. Fix a node v. Let P_v denote a path taken by the message from r to arrive at v in the schedule S. Let E_t denote the set of all directed edges in $\cup_v P_v$ that are activated in round t in S. By definition of the telephone model, for distinct (u_1, v_1) and (u_2, v_2) in E_t, $u_1 \neq u_2$ and $v_1 \neq v_2$. Furthermore, by our choice of the paths, we obtain that the edges of P_v appear in order of increasing time in the collection of E_ts.

We now argue that a reverse schedule in which the activated sets are given by $E'_t = \text{REV}(E_{T-t})$ forms a schedule for gathering in the edge-star model. In any round t, for any distinct (u_1, v_1) and (u_2, v_2) in E_t, we have $u_1 \neq u_2$ and $v_1 \neq v_2$; therefore, $\text{REV}(E_t)$ is a matching, and is a valid set of edges to activate in the edge-star model in round $T - t$. Since the edges of P_v appear in order of increasing time in the collection of E_ts, the edges of the $\text{REV}(P_t)$ appear in order of increasing time in the collection of E'_ts. Consequently, the message from any node v is delivered to the root in T rounds.

Once the root has all the messages, we can complete the gossip by running the broadcast schedule. Since any schedule in the telephone model is valid in the edge-star model, it follows that this broadcast completes in T rounds. We thus have a gossip schedule that completes in the edge-star model in $2T$ rounds. ◀

There exists an $O(\frac{\log n}{\log \log n})$ approximation for telephone broadcast [7]. Therefore this same approximation holds for the edge-star gossip problem.

3.2 Multicommodity multicast on a tree

In this part, we consider the multicommodity multicast problem in the edge-star model in the special case where our host graph is a tree. Here we give a reduction to telephone broadcast. When the host graph is a tree, the path taken by any message is known, so we simply need to coordinate the communications.

▶ **Lemma 3.3.** *There is an $O(\frac{\log n}{\log \log n})$ approximation for the edge-star multicommodity multicast problem in a tree.*

Proof. We will start by choosing some vertex r to be the root of the tree. Let the optimal solution take time D (we can try all $2n$ possible values for D only losing a polynomial factor in runtime). Now for each demand pair, (s_i, t_i) the message will have to go from s_i to $lca(s_i, t_i)$, and then from the $lca(s_i, t_i)$ down to t_i. Bringing all the messages down the tree from $lca(s_i, t_i)$ to t_i can be done in time $D+1$; we spend $D+1$ time steps alternating between the odd layers broadcasting their messages down and the even layers broadcasting their message down. Since each layer is a collection of edge-disjoint stars, this can be implemented in one round in the edge-star model.

The hard part is bringing the messages up from s_i to $t'_i = lca(s_i, t_i)$. So, we will consider that we simply have the constraints of the form (s_i, t'_i). First we will break the tree up into sets of $2D$ consecutive layers starting every D layers. This guarantees that every constraint (s_i, t_i) is in some set of $2D$ layers.

Now consider some $2D$ layers in the tree. Look at the union of all the (s_i, t'_i) paths in these layers. These form a forest, where each tree has depth at most $2D$ and each node has a max degree of D; so each tree has poise $5D$ (recall that the poise is the sum of the maximum degree and the diameter). Therefore each of these trees can gather all their messages to their uppermost nodes in time $O(D\frac{\log n}{\log\log n})$.

We can run all the gathers to satisfy (s_i, t'_i) in two groups; we can run every other set of $2D$ layers in the tree simultaneously, as they are disjoint. Hence, in time $O(D\frac{\log n}{\log\log n})$, we can satisfy the demands (s_i, t'_i). After this, in $D+1$ more steps, we can satisfy the demands (t'_i, t_i). Therefore in time $O(D\frac{\log n}{\log\log n})$ we satisfy all the (s_i, t_i) demands. ◀

3.3 Symmetric Multicommmodity Multicast

Note that the symmetric multicommodity multicast problem in the telephone model is equivalent (within constant factors) to the general multicommodity multicast problem [14, 3] for which an $\tilde{O}(2^{\sqrt{\log k}})$ approximation algorithm is known, where k is the number of terminals [13]. We show a reduction from the symmetric multicommodity multicast problem in the edge-star model to the symmetric multicommodity multicast problem in the telephone model, losing an additional $O(\frac{\log^3 n}{\log\log n})$ factor in the approximation ratio in an n-node graph.

▶ **Theorem 3.4.** *Given a ρ-approximation for the symmetric multicommodity multicast problem on k terminal pairs in an n-node undirected graph under the telephone model, we can design an $O(\rho \cdot \log^2 k \cdot \frac{\log n}{\log\log n})$ approximation for the same problem in the edge-star model.*

Proof. Given an optimal solution to symmetric multicommodity multicast in the edge-star model, we demonstrate a solution to the symmetric multicommodity multicast problem in the telephone model with a poly-log multiplicative loss in performance. Consider an input instance with demand pairs $\{s_i, t_i\}$ for $i = 1 \cdots k$ on an undirected graph G. Consider an optimal schedule for the edge-star symmetric multicommodity multicast problem on this instance. This defines for each pair s, t, a pair of paths from one node to the other where the edges of the paths are labeled in increasing time order denoting the periods in which these edges participated in an information transmission. Suppose the optimal time for multicasting is L; then these paths are of length at most L. Also, given the in-degree one bound for the edge-star model (each receiver can listen to at most one transmitter in this wireless model), the indegree of the sugraph representing the union of these optimal transmissions is also at most L. Our goal is to use these paths to aggregate the messages from a set of these pairs into a subset of carefully selected terminals using a reverse broadcast scheme, and then transmit the aggregated messages back to the corresponding mates of these sources. Both

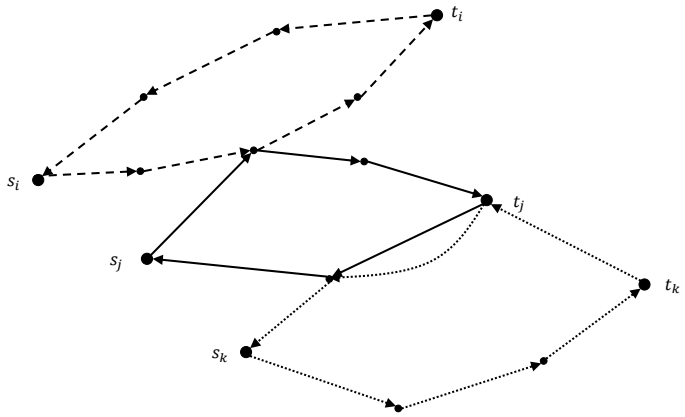


Figure 2 Here is an example of the optimal paths between some (s_i, t_i) pairs. Here we see that the (s_j, t_j) pair intersects (s_i, t_i) and (s_k, t_k), but (s_i, t_i) and (s_k, t_k) do not intersect.

these steps of gathering and sending will be accomplished using multicommodity multicast instances in the telephone model.

To define the aggregation pattern, define an auxiliary graph H with one node per *demand pair* s_i, t_i. This graph is only for the sake of argument so we will use optimal paths in the edge-star multicommodity multicast scheme in defining it. Note that the optimal transmission paths for a pair represent two paths: one from s_i to t_i and the second from t_i to s_i, where these two paths may share edges. Concatenated together they define what we will call an "optimal cycle" for this pair. Define an edge between two pairs if their optimal cycles intersect at a node. In Figure 2, we can see an example of when optimal cycles intersect. Thus H defines the conflict or interference between the demand pairs in the optimal multicommodity multicast schedule in the edge-star model.

We now use a network decomposition procedure [2] on H to decompose the k demand pairs into $\log_2 k$ disjoint layers with the following property: the set of nodes in each layer can be decomposed into node-disjoint shallow trees, i.e., each tree in one of the layers has diameter at most $2\log_2 k$. This decomposition is done as follows: pick any vertex v in H and build a BFS tree from v. Now let i be the smallest depth such that the number of nodes at depth i or less is more than the number of nodes at depth $i+1$. Put v and everything within distance i of v into the current layer. Now remove v and it's BFS tree up to depth $i+1$ from H. Repeat this process to form each layer. Once H is empty, let U be the vertices not yet assigned to a layer. Then start forming a new layer from the graph $H[U]$.

This process assigns at least half of the remaining nodes to the current layer, hence we build at most $\log_2 k$ layers. The diameter of each component in a layer is at most $2\log_2 k$, because as we move down the BFS tree the number of nodes contained in it double at each step.

Now we can use these layers to define our gathering problems. Consider one layer i and one tree $T_{i,j}$ in this layer in the decomposition. This represents a shallow subgraph in H, so let us root this at a demand pair denoted P_{ij}. By following the paths in this subgraph from every other node to P_{ij}, we can replace their intersections with paths in the optimal multicast originating at each terminal s in any of the pairs to one of the two terminals, say

t_{ij} in the pair P_{ij}. This defines one of the gathering trees gathering to the terminal t_{ij}. By construction, the in-degree of any node in the gathering tree is at most L and the distance from any node to the root t_{ij} is at most $O(L \log k)$. Note that by the disjointness of the subgraphs in one layer i, all the gather trees are node disjoint. For each gather tree T_{ij}, we now set up a gathering multicast problem with all the terminals in the tree going to the root t_{ij}. Note that since each tree has total degree + diameter at most $O(L \log k)$, the poise of each tree is bounded by $O(L \log k)$ and thus each of these trees has a gathering schedule in the telephone model taking at most $O(Poise \cdot \frac{\log n}{\log \log n})$ steps in an n-node graph [14]. This gives a feasible solution to the set of all gathering problems in one layer i running in time $O(L \cdot \log k \cdot \frac{\log n}{\log \log n})$. Repeating this over the layers finally gives a set of gathering problems in the telephone model that complete in total time $O(L \cdot \log^2 k \cdot \frac{\log n}{\log \log n})$.

Note that the same schedules can be reversed to send all the gathered information in each tree to all the terminals in a tree finishing the requirements. Employing a ρ-approximation for this multicommodity multicast problem in the telephone model proves the theorem. ◀

3.4 Asymmetric Multicommodity Multicast

For the edge-star asymmetric multicommodity multicast problem, we will use the network decomposition used in the previous proof, along with telephone broadcast in trees with small poise.

▶ **Theorem 3.5.** *There is an $\tilde{O}(n^{\frac{2}{3}})$-approximation for the asymmetric multicommodity multicast problem in the edge-star model.*

Proof. We develop the algorithm in two phases. First, we design an $\tilde{O}(\sqrt{p})$-approximation algorithm for the case with p demand pairs (note that p can be up to $O(n^2)$ in an n-node graph). Then we combine this with an algorithm that satisfies all the demands in the in-neighborhood of a node in the demand graph with high indegree to get the final result.

A Greedy Algorithm. To design the $\tilde{O}(\sqrt{p})$-approximation algorithm, we use a greedy method: assume that the value of the optimal multicast time is L (we can try all the $2n$ possible guesses in parallel to dispense this assumption with a polynomial running-time overhead). For every unsatisfied demand pair (s_i, t_i) (note that demand pairs are ordered in the asymmetric case), we look for a path of length at most L from s_i to t_i. If we find one, we add it to the greedy collection and delete all the nodes in this path. Suppose we are able to collect g paths for the pairs denoted G in the greedy phase until we can find no more paths of small length for the remaining demands.

Now it must be the case that all optimal paths for the remaining demands in $P \setminus G$ must intersect the greedy paths. This implies that for every demand pair (s, t) in $P \setminus G$, we can follow its optimal path to its intersection with one of the greedy paths, say for the pair (s_i, t_i), and then continue in the greedy path to t_i. In this way, every demand source in $P \setminus G$ can be routed and assigned to one of the sinks in the greedy pairs G in a collection of paths: each such path has length at most $2L$ (coming from at most L steps to the intersection with the greedy collection and another L from the intersection to the sink at the end of this greedy path); also the indegree of the collection of these paths is at most $L + 1$ since they arise from the optimal collection plus the greedy subgraph which adds at most one to each node's indegree. We now set up a dummy broadcast problem (following Nikzad and Ravi [13]) by hooking up the set of sinks at the end of the greedy paths, say $T(G)$, as leaves in a complete binary tree with new dummy nodes and a dummy root t. We solve for the broadcast problem in this graph from the dummy root t to all the sources s_i in all the pairs.

By the above construction, there exists a tree of poise $O(L + \log n)$ that connects all the sources to this root. From the result of Ravi [14], this implies a broadcast scheme completing in $O(L\frac{\log n}{\log\log n})$. Using an α-approximation algorithm for broadcast in the telephone model, we get a tree that assign the sources in P to the sinks in $T(G)$ in $O(\alpha \cdot L\frac{\log n}{\log\log n})$ steps. Let us denote the set of sinks in $T(G)$ by $t'_1, \cdot, t'_g$ and the set of sources assigned to a sink t'_i by S_i.

The remaining task is to send back the messages gathered from S_i at t'_i to the sinks corresponding to the sources in S_i - let us denote this sink set by T_i. Note that by construction, all the sinks in T_i are at a distance at most $O(\alpha \cdot L\frac{\log n}{\log\log n})$ from t'_i by following the paths to the corresponding source s and then concatenating the undirected path to its mate t. However, these local broadcasts must obey the edge-subgraph condition of having indegree at most one which is tricky to enforce.

If the number of greedy pairs $g = |G|$ is at least $\sqrt{p}$, we simply satisfy these pairs and move to the next iteration: the number of such iterations is at most $\sqrt{p}$ and each iteration can be implemented in $O(L)$ steps (running the disjoint greedy path schedules in parallel). If the number of pair is less than $\sqrt{p}$, we can carry out the broadcast from each greedy sink t'_i to its sink set T_i in time $O(\alpha \cdot L\frac{\log n}{\log\log n})$ by reversing the gathering in the earlier broadcast tree and extending it to the corresponding sinks. Processing these trees one after another, we use a total of $O(\sqrt{p} \cdot \alpha \cdot L\frac{\log n}{\log\log n})$. Since α is sublogarithmic [6], we finally get an $\tilde{O}(\sqrt{p})$-approximation as claimed.

A Local Algorithm. For the second ingredient we observe that if the in-degree of any node v in the demand graph is δ, then we can satisfy all the demand requirements of the predecessors of v in the demand graph $In(v)$ in time $\tilde{O}(L)$. Note that since all the terminals in $In(v)$ send their message to v, the union of the directed paths that transmit these messages in the optimal solution have distance at most L from the terminals to v and induce an in-degree of at most L. This defines a tree of poise $O(L)$ and hence enables us to find a broadcast scheme that gathers all the messages from $In(v)$ at v in time $\tilde{O}(L)$. By reversing this broadcast tree and then following the optimal paths from each terminal in $In(v)$ to its other sinks, we can find a tree of depth (not poise) at most $\tilde{O}(L)$ rooted at v where these messages are gathered. Since v is the only node sending out the gathered messages, we can send all these messages to their intended sinks in a breadth-first tree in time $\tilde{O}(L)$ in the edge-star model. Note that we have taken care of all the demands originating in $|In(v)|$ nodes.

Combining the two algorithms. We can now combine the two algorithms as follows: As long as p, the number of demand pairs in the n-node graph, is at least $\Omega(n^{\frac{4}{3}})$, we use the local algorithm. By averaging over the indegrees that partition the demand pairs, there exists a node of indegree at least $\Omega(n^{\frac{1}{3}})$ in the demand graph. The local algorithm thus satisfies the demands originating in at least this many nodes in one iteration. The number of iterations is thus at most $n^{\frac{2}{3}}$ each taking $\tilde{O}(L)$ multicast steps. On the other hand, when p drops below $O(n^{\frac{4}{3}})$, we use the greedy algorithm to get an approximation ratio of $\tilde{O}(\sqrt{p}) = \tilde{O}(n^{\frac{2}{3}})$ giving the result. ◀

4 Conclusion

We have obtained new results in the approximability of rumor spreading problems in the well-studied radio model as well as a new model motivated by wireless communications, which we call the edge-star model. For the radio model, we present an $\Omega(n^{1/2-\epsilon})$ hardness of

approximation bound for radio gossip, making progress on an open problem mentioned in [11]. For the edge-star model, we present an $O(\frac{\log n}{\log \log n})$ approximation algorithm for gossip, an $\tilde{O}(2^{\sqrt{\log n}})$ approximation algorithm for symmetric multicommodity multicast, and an $\tilde{O}(n^{2/3})$ approximation algorithm for asymmetric multicommodity multicast. Our approximation algorithms expose relationships between the edge-star model and the well-studied telephone model.

Our work leaves several interesting open problems. Among the nine cells listed in the matrix of Table 1 of Section 1, only radio broadcast and edge-star broadcast are resolved. Significant gaps between the best known upper and lower bounds on approximability remain for telephone broadcast, the gossip problem under all three models, and the multicommodity multicast problem under all three models. In the edge-star model, the symmetric and asymmetric versions of the multicommodity multicast problem are distinct, and both are open, in terms of the best approximation factor achievable in polynomial-time.

References

1 N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A lower bound for radio broadcast. *J. Comput. Syst. Sci.*, 43:290–298, 1991.

2 B. Awerbuch and D. Peleg. Sparse partitions. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 503–513. IEEE, 1990.

3 A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Multicasting in heterogeneous networks. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 448–453, 1998.

4 M. Elkin and G. Kortsarz. A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem. *SIAM J. Comput.*, 35(3):672—-689, 2005.

5 M. Elkin and G. Kortsarz. Polylogarithmic additive inapproximability of the radio broadcast problem. *SIAM J. Discrete Math*, 19(4):881—-899, 2005.

6 M. Elkin and G. Kortsarz. Sublogarithmic approximation for telephone multicast. *J. Comput. Syst. Sci.*, 72(4):648—-659, 2006.

7 M. Elkin and G. Kortsarz. An improved algorithm for radio broadcast. *ACM Transactions on Algorithms (TALG)*, 3(1):8, 2007.

8 U. Feige and J. Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57:187–199, 1998.

9 Rajiv Gandhi, Magnús M Halldórsson, Christian Konrad, Guy Kortsarz, and Hoon Oh. Radio aggregation scheduling. In *Proceedings of the 11th International Symposium on Algorithms and Experiements for Wireless Sensor Networks (ALGOSENSORS)*, 2015.

10 L. Gasieniec, D. Peleg, and Q. Xin. Faster communication in known topology radio networks. *Distributed Computing*, 19(4):289–300, 2007.

11 L. Gąsieniec. On efficient gossiping in radio networks. In Shay Kutten and Janez Žerovnik, editors, *Structural Information and Communication Complexity*, volume 5869 of *Lecture Notes in Computer Science*, pages 2–14. Springer Berlin Heidelberg, 2010.

12 D. R. Kowalski and A. Pelc. Optimal deterministic broadcasting in known topology radio networks. *Distributed Computing*, 19(3):185–195, 2007.

13 A. Nikzad and R Ravi. Sending secrets swiftly: Approximation algorithms for generalized multicast problems. In *Automata, Languages, and Programming*, pages 568–607. Springer, 2014.

14 R Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 202–213. IEEE, 1994.

The Price of Local Power Control in Wireless Scheduling*

Magnús M. Halldórsson and Tigran Tonoyan

ICE-TCS, Reykjavik University, Iceland
{magnusmh,ttonoyan}@gmail.com

Abstract

We consider the problem of scheduling wireless links in the physical model, where we seek an assignment of power levels and a partition of the given set of links into the minimum number of subsets satisfying the signal-to-interference-and-noise-ratio (SINR) constraints. Specifically, we are interested in the efficiency of local power assignment schemes, or *oblivious power schemes*, in approximating wireless scheduling. Oblivious power schemes are motivated by networking scenarios when power levels must be decided in advance, and not as part of the scheduling computation.

We present the first $O(\log\log\Delta)$-approximation algorithm, which is known to be best possible (in terms of Δ) for oblivious power schemes, where Δ is the longest to shortest link length ratio. We achieve this by representing interference by a conflict graph, which allows the application of graph-theoretic results for a variety of related problems, including the weighted capacity problem. We explore further the contours of approximability and find the choice of power assignment matters; that not all metric spaces are equal; and that the presence of weak links makes the problem harder. Combined, our results resolve the *price of local power* for wireless scheduling, or the value of allowing unfettered power control.

1998 ACM Subject Classification C.2.1 Network Architecture and Design; Wireless communication

Keywords and phrases Wireless Scheduling, Physical Model, Oblivious Power

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.529

1 Introduction

We treat the fundamental *scheduling* problem of partitioning a given set of transmission requests (links) in a wireless network into the fewest possible feasible subsets. Scheduling problems arise from the MAC layer of wireless networks, and solving them requires effective spatial reuse while dealing with wireless interference. We use the *SINR model* for modeling interference, where signal decays as it travels and a transmission is successful if its strength at the receiver exceeds the accumulated signal strength of interfering transmissions by a sufficient factor. Although the standard analytic assumption that signal decays polynomially with the distance traveled is far from realistic [36, 31], it has been shown that results obtained with that assumption can be translated to the setting of arbitrary measured signal decay [2, 13]. A number of studies dedicated to elucidating algorithmic properties of the SINR model has appeared in recent years (e.g., [32, 11, 1, 6, 29, 27, 21, 5, 25]).

Abstracting wireless interference by *conflict graphs* is much more common in wireless research. Arbitrary graphs are too general to be useful (in the worst case) for scheduling

* The work is supported by grant-of-excellence 120032011 and grant 152679-051 from the Icelandic Research Fund.

problems. Instead, the standard modus operandi is to assume geometric intersection graphs, where nodes are represented by disks, such as *unit disk graphs* or the *protocol model* [14]. Unfortunately, disk graphs provably lack fidelity to the reality of wireless signals, being simultaneously too conservative and too loose [34, 33]. Yet, the graph abstraction is cleaner and connects better to the literature, which leads us to search for alternative classes.

Problem Formulations. Given as input is a set Γ of n communication *links*; each link is a pair of a sender and receiver, which are nodes in a metric space. The senders can adjust their power as needed; in each of the problems we consider, finding the appropriate power assignment is a part of the problem. A subset $S \subseteq \Gamma$ of links is *feasible* if there exists a power assignment for which the transmission on each link satisfies the SINR formula (see Section 2) when the links in S transmit simultaneously. We treat the following two problems:

Scheduling: Partition Γ into fewest number of feasible sets.

WCapacity: Find the maximum weight feasible subset of Γ, when the links have positive weights. When WCapacity is restricted to unit weights, we get the related unweighted Capacity problem. The WCapacity problem is of fundamental importance to dynamic scheduling where requests arrive over time, as it is demonstrated in a celebrated paper [37].

Optimal solutions to our problems may require global power management: the power assigned to a link may depend on all the other links. However, it is desirable in various restricted settings to let the power chosen for a link depend only on the link itself, specifically on the link length. Such local power regimes are called *oblivious*. The use of oblivious power assignments may be forced in cases when having a separate power control phase is not affordable. Oblivious powers may also be preferable due to scalability concerns: when a new link is added, the power assignments of other links are left intact (as opposed to globally managing power).

The main goal of this paper is to solve the Scheduling and WCapacity problems using *only* oblivious power assignments, while still comparing the quality of the solution to the optimal solution that can use arbitrary power assignment. It was shown in [8] that every oblivious power assignment can be worst possible factor n from optimal. In terms of the parameter Δ, the bound becomes $\Omega(\log\log\Delta)$ [8, 15], i.e. the factor n may appear only when the network contains exponentially long links. Indeed, there is an algorithm using oblivious power that achieves $O(\log\log\Delta\log n)$-approximation (compared with best achievable with arbitrary power) [15]. For Capacity, this was improved to $O(\log\log\Delta)$-factor [18], which holds for every metric space and for a wide range of oblivious power assignments.

Finding constant-factor approximations for Scheduling is still an open problem. However, there are several approaches giving logarithmic approximation. The (unweighted) Capacity problem has an efficient constant-factor approximation, due to Kesselheim [27] that holds for general metrics [28], as well as for versions with various fixed power assignments [19]. This immediately yields $O(\log n)$-approximation for Scheduling. A different approach is to divide the links into groups of nearly equal length and schedule each group separately. Following this approach, numerous $O(\log\Delta)$-approximation results have been argued [12, 10, 15], where Δ is the ratio between the longest and shortest link length. In a recent work, we propose a novel conflict-graph based approach that yields $O(\log^*\Delta)$ approximation for Scheduling and WCapacity using *non-oblivious* power assignment [22].

Results. Our main result is $O(\log\log\Delta)$-approximation algorithms for Scheduling and WCapacity using oblivious power assignments. This is an exponential improvement over existing approximations using oblivious powers and matches the known lower bounds [8, 15].

This is also the first improvement over logarithmic approximations for fixed power scheduling, where we compare to the optimum schedule with respect to *given* power assignment. Even though $O(\log\log\Delta)$ is weaker than the approximation of $O(\log^*\Delta)$ obtained in [22], it still demonstrates the remarkable closeness of oblivious powers to the optimum power assignments. After all, $\log\log\Delta \leq 5$ in all practical applications.

Unlike the state of affairs for the Capacity problem, our results are surprisingly sensitive to the metric and the exact power assignment. They hold for doubling metrics, but provably fail in general (or even tree) metrics, and they hold for a range of power assignments, while for others they provably fail (including the most common assignments – *uniform* and *linear*).

Our main result is obtained by using the conflict graph framework of [22], which essentially reduces the notoriously hard SINR optimization to graph problems, for which a large body of theory can be brought to bear.

Further Related Work. Gupta and Kumar [14] proposed the geometric version of SINR and initiated average-case analysis of capacity known as scaling laws. Moscibroda and Wattenhofer [32] initiated worst-case analysis in the SINR model. Early work on the Scheduling problem includes [4, 7, 3]. NP-completeness has been shown for Scheduling with different forms of power control: none [12], limited [26], and unbounded [30]. Distributed algorithms attaining $O(\log n)$-approximation are also known [29, 16]. Scheduling and WCapacity have also been considered for *fixed oblivious power assignments* [23, 8, 15, 20, 9]. The only known constant-factor approximation algorithms for these problems are obtained in the case of the *linear power scheme* [20, 39].

2 Model and Definitions

Communication Links. Consider a set Γ of n *links*, numbered from 1 to n. Each link i represents a unit-demand communication request from a sender s_i to a receiver r_i - point-size wireless transmitter/receivers located in a metric space with distance function d. We denote $d_{ij} = d(s_i, r_j)$ the distance from the sender of link i to the receiver of link j, $l_i = d(s_i, r_i)$ the *length* of link i and $d(i,j) = d(j,i)$ the minimum distance between a node of link i and a node of link j. We let $\Delta(\Gamma)$ denote the ratio between the longest and shortest link lengths in Γ, and drop Γ when clear from context. A set of links S is *equilength* if $\Delta(S) \leq 2$.

Power Schemes. A *power assignment* for Γ is a function $P : \Gamma \to \mathbb{R}_+$. For each link i, $P(i)$ defines the power level used by the sender node s_i. We will be particularly interested in *power schemes* P_τ of the form $P_\tau(i) = C \cdot l_i^{\tau\alpha}$, where C is constant for the given network instance. These are called *oblivious power assignments* because the power level of each link depends only on a local information - the link length. Examples of such power schemes are *uniform* power scheme (P_0), *linear* power scheme (P_1) and *mean* power scheme ($P_{1/2}$) [8].

SINR Feasibility. In the *physical model (or SINR model)* of communication [35], a transmission of a link i is successful if and only if

$$\mathcal{S}_i \geq \beta \cdot \left(\sum_{j \in S \setminus \{i\}} \mathcal{I}_{ji} + N \right), \tag{1}$$

where $\mathcal{S}_i$ denotes the received signal power of link i, $\mathcal{I}_{ji}$ denotes the interference power on link i caused by link j, $N \geq 0$ is a constant denoting the ambient noise, $\beta > 0$ is the minimum

SINR (Signal to Interference and Noise Ratio) required for a message to be successfully received and S is the set of links transmitting concurrently with link i. If P is the power assignment used, then $\mathcal{S}_i = \frac{P(i)}{l_i^\alpha}$ and $\mathcal{I}_{ji} = \frac{P(j)}{d_{ji}^\alpha}$, where $\alpha \in (2, 6)$ is the path-loss exponent.

A set L of links is called *P-feasible* if the condition (1) holds for each link $i \in L$ when using power assignment P. We say L is *feasible* if there exists a power assignment P for which L is P-feasible. Similarly, a collection of sets is P-feasible/feasible if each set in the collection is.

Capacity and Scheduling Problems. Scheduling denotes the problem of partitioning a given set Γ into the minimum number of feasible subsets (or *slots*). WCapacity denotes the problem where we are also given a weight function $\omega : \Gamma \to \mathbb{R}_+$ on the links and we seek a maximum weight feasible subset S of Γ.

Affectance. Following [23], we define the *affectance* $a_P(i, j)$ of link i by link j under power assignment P by

$$a_P(j, i) = c_i \frac{\mathcal{I}_{ji}}{\mathcal{S}_i} = c_i \frac{P(j) l_i^\alpha}{P(i) d_{ji}^\alpha},$$

where $c_i = 1/(1 - \beta N l_i^\alpha / P(i))$ is a factor depending on the properties of link i[1]. We let $a_P(j, j) = 0$ and extend a_P additively over sets: $a_P(S, i) = \sum_{j \in S} a_P(j, i)$ and $a_P(i, S) = \sum_{j \in S} a_P(i, j)$. It is readily verified that a set of links S is feasible if and only if $a_P(S, i) \leq 1/\beta$ for all $i \in S$. We call a set of links *p-P-feasible* for a parameter $p > 0$ if $a_P(S, i) \leq 1/p$.

The following is an important tool showing that modifying the threshold value β by a constant factor affects the solutions only by a constant factor.

▶ **Theorem 1** ([17]). *Any p-P-feasible set can be partitioned into $\lceil 2p'/p \rceil$ subsets, each of which is p'-P-feasible.*

We make the standard assumption that for all links i in the instance, received signal power is a little higher than necessary to overcome the noise term N alone in the absence of any other transmission: $P(i) \geq c\beta N l_i^\alpha$ for some constant $c > 1$. This can be achieved by scaling the power levels of links. This assumption helps to avoid the terms c_i in the affectance formula. Indeed, it implies that $c_i \leq c/(c-1)$ for all i. Then given e.g. a Scheduling instance Γ, we can solve it with $c_i = 1$ for all i and $\beta' = (c-1)\beta/c$, getting a feasible solution for the original problem. Moreover, by Thm. 1, the number of slots obtained will be at most a constant factor away from the optimum of the original problem. Thus, we assume henceforth that $c_i = 1$ for all links i, i.e. $a_P(i, j) = \frac{P(j) l_i^\alpha}{P(i) d_{ji}^\alpha}$. We have in particular that $a_{P_\tau}(i, j) = l_i^{(1-\tau)\alpha} \cdot l_j^{\tau\alpha} / d_{ij}^\alpha$.

Remark. In practice, there is an upper limit P_{max} on the available power level of links and for some links, even setting $P(i) = P_{max}$ can be insufficient for having $P(i) \geq c\beta N l_i^\alpha$. Such links are called *weak links*. Our assumption thus amounts to excluding weak links. Weak links are further discussed in Sec. 5.

[1] If the denominator of c_i is 0, i.e. $P(i) = \beta N l_i^\alpha$, then link i must always be scheduled separately from all other links. We assume that there are no such links.

Fading Metrics. The *doubling dimension* of a metric space is the infimum of all numbers $\delta > 0$ such that every ball of radius $r > 0$ has at most $C\epsilon^{-\delta}$ points of mutual distance at least ϵr where $C \geq 1$ is an absolute constant, $\delta > 0$ and $0 < \epsilon \leq 1$. Metrics with finite doubling dimension are called *doubling*. For instance, the m-dimensional Euclidean space is doubling with doubling dimension m [24]. We will assume for the rest of the paper that the links are located in a metric space with doubling dimension $m < \alpha$. Such metrics are called *fading metrics*.

3 Conflict Graphs and Oblivious Power Scheduling

Conflict graphs are graphs defined over the set of links. Let us call a conflict graph $A(L)$ an *upper bound graph* for a set L, if there is a power scheme P_τ such that each independent set in $A(L)$ is P_τ-feasible. Similarly, we call a graph $B(L)$ a *lower bound graph* for L if there is a power scheme $P_{\tau'}$ such that each $P_{\tau'}$-feasible set induces an independent set in $B(L)$. Note that the chromatic numbers of $A(L)$ and $B(L)$ give upper and lower bounds for Scheduling with oblivious power schemes. Moreover, if the vertex coloring problem for $A(L)$ can be efficiently approximated, then the upper bound is constructive. Now, our aim is to construct upper and lower bound graphs with $\tau = \tau'$ such that the gap between their chromatic numbers is bounded. The less the gap, the better colorings of $A(L)$ approximate Scheduling with oblivious power P_τ.

The outline of this section is as follows. First, we present a family of conflict graphs introduced in [22] and point out a sub-family $\mathcal{G}_\gamma$ that are lower bound graphs. Next, we present a family of upper bound graphs $\mathcal{G}^\delta_\gamma$ and show that the gap between the chromatic numbers is $O(\log\log\Delta)$. This section is concluded with the main theorem which, based on the method outlined above, presents $O(\log\log\Delta)$-approximation algorithms for Scheduling and WCapacity with oblivious power schemes.

Conflict Graphs, Lower Bound Graphs. Let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be a positive monotonically non-decreasing function. Two links i, j are *f-independent* if

$$\frac{d(i,j)}{l_{min}} > f\left(\frac{l_{max}}{l_{min}}\right),$$

where $l_{min} = \min\{l_i, l_j\}, l_{max} = \max\{l_i, l_j\}$, and otherwise they are *f-conflicting*. A set of links is f-independent if they are pairwise f-independent.

Given a set L of links, $\mathcal{G}_f(L)$ denotes the graph with vertex set L where two vertices $i, j \in L$ are adjacent if and only if they are f-adjacent.

We will be particularly interested in conflict graphs $\mathcal{G}_f$ with $f(x) \equiv \gamma$ and $f(x) = \gamma \cdot x^\delta$ for constants $\gamma > 0$ and $\delta \in (0, 1)$. We will use the notation $\mathcal{G}_\gamma$ in the former case and the notation $\mathcal{G}^\delta_\gamma$ in the latter case. We will refer to independence (conflict) in $\mathcal{G}_\gamma$ as γ-independence (γ-conflict, resp.) and to independence (conflict) in $\mathcal{G}^\delta_\gamma$ as (γ, δ)-independence ((γ, δ)-conflict, resp.). Note that $\mathcal{G}_\gamma$ is equivalent to $\mathcal{G}^0_\gamma$.

It will be useful to note that two links i, j are γ-independent iff $d(i, j) > \gamma l_j$ and are (γ, δ)-independent iff $d(i, j) > \gamma l_i^\delta l_j^{1-\delta}$, where l_i is the longer link, i.e. $l_j \leq l_i$.

We will need the following properties of conflict graphs $\mathcal{G}^\delta_\gamma$ that are obtained by applying the results of [22] to our conflict graphs (i.e. $\mathcal{G}_f$ with $f(x) = \gamma x^\delta$). These properties hold for any set L of links in a metric space of fixed doubling dimension. We let $\chi(G)$ denote the chromatic number of a graph G.

▶ **Theorem 2.** *Let $\delta \in (0,1)$ be constant. It holds that*
1. *for any constant $\gamma > 0$, $\chi(\mathcal{G}_\gamma^\delta(L)) = \chi(\mathcal{G}_\gamma(L)) \cdot O(\log\log\Delta)$,*
2. *if $\beta > 1$, there is a constant $\gamma > 0$ s.t. $\mathcal{G}_\gamma(L)$ is a lower bound graph,*
3. *vertex coloring and maximum weighted independent set problems are constant factor approximable in graphs $\mathcal{G}_\gamma^\delta(L)$.*

The first property ([22, Thm. 1]) bounds the gap between graphs $\mathcal{G}_\gamma^\delta(L)$ and $\mathcal{G}_\gamma(L)$. The second property ([22, Thm. 4]) shows that for an appropriate constant γ, the graph $\mathcal{G}_\gamma$ is a lower bound graph, i.e. each feasible set is independent in $\mathcal{G}_\gamma$ (this includes also P_τ-feasible sets for *any* τ). The last property ([22, Prop. 1]) shows that the graphs $\mathcal{G}_\gamma^\delta$ are algorithmically accessible.

Upper Bound Graphs. Here we show that for appropriate values of δ and γ, graphs $\mathcal{G}_\gamma^\delta(L)$ are upper bound graphs, i.e. each independent set in $\mathcal{G}_\gamma^\delta(L)$ is feasible with the appropriate oblivious power assignment. This complements the conflict graph framework described in the beginning of the section.

In order to bound the affectance of a given link i by an independent set S of links, we first split S into length classes (i.e. equilength subsets) and bound the affectance of i by each length class separately (lemmas 6 and 7). Then we combine the obtained bounds in a series that converges under the assumption that the links are in a fading metric (Cor. 8). The affectance of link i by each length class is bounded by using the common "concentric annuli" argument where the rough idea is to partition the metric space into concentric annuli centered at an endpoint of i, bound the number of links in each annulus using link independence and the doubling property of the space, use these bounds to bound the affectance by links from different annuli and combine them into a converging series. This scheme has also been used in [22]. The main difficulty here is that we have to deal with the affectance of a given link by both longer and shorter links, as opposed to [22] where we had to consider only shorter links. The parameter δ of $\mathcal{G}_\gamma^\delta$ has to be chosen very carefully in order to guarantee bounded affectance by both longer and shorter links.

We will obtain a slightly stronger result than feasibility. Our results hold in terms of the function $f_\tau(i,j)$ with a parameter $\tau \in [0,1]$, where

$$f_\tau(i,j) = \frac{l_i^{\tau\alpha} \cdot l_j^{(1-\tau)\alpha}}{d(i,j)^\alpha}.$$

Note that for any pair of links i, j, $a_{P_\tau}(i,j) \leq f_\tau(i,j)$. The function $f_\tau(i,j)$ is extended additively to sets of links, similar to the function $a_P(i,j)$.

In the following core lemma we show that the affectance of a fixed link i by an independent equilength set S of links (i.e. $\Delta(S) \leq 2$) can be bounded by the ratio of the length l_i and the minimum length in S if S and i are not too close to each other. The idea of the proof is the "concentric annuli" argument described above. We will use the following two facts.

▶ **Fact 3.** *Let $\alpha \geq 1$ and $r \geq 0$ be real numbers. Then $\frac{1}{r^\alpha} - \frac{1}{(r+1)^\alpha} \leq \frac{\alpha}{(r+1)^{\alpha+1}}$.*

▶ **Fact 4.** *Let $g(x) = \dfrac{1}{(q+x)^\gamma}$, where $\gamma > 1$ and $q > 0$. Then $\sum_{r=0}^{\infty} g(r) \in O\left(\dfrac{1}{q^{\gamma-1}} + \dfrac{1}{q^\gamma}\right)$.*

▶ **Lemma 5.** *Let $\delta, \tau \in (0,1)$ and $\gamma \geq 1$, let S be an equilength set of 1-independent links, and let i be a link s.t. i, j are (γ,δ)-independent for all $j \in S$ and either $l_i \geq l_j$ for all $j \in S$ or $l_i \leq l_j$ for all $j \in S$. Then,*

$$f_\tau(S,i) \in O\left(\gamma^{m-\alpha}\left(\frac{l_i}{\ell}\right)^{(1-\tau)\alpha-\delta'(\alpha-m)} \cdot \min\left\{1, \frac{l_i}{\ell}\right\}^{-\delta'}\right),$$

where ℓ denotes the shortest link length in S and $\delta' = \delta$ if $l_i \geq \ell$ and $\delta' = 1-\delta$ otherwise.

Proof. First, let us split S into two subsets S' and S'' such that S' contains the links of S that are closer to r_i than to s_i, i.e. $S' = \{j \in S : \min\{d(s_j, r_i), d(r_j, r_i)\} \leq \min\{d(s_j, s_i), d(r_j, s_i)\}\}$ and $S'' = S \setminus S'$. Let us consider the set S' first.

For each link $j \in S'$, let p_j denote the endpoint of j that is closest to node r_i. Denote $q = \gamma l_i^{\delta'} \ell^{-\delta'}$. Consider the subsets $S_1, S_2, \ldots$ of S', where $S_r = \{j \in S' : d(j,i) = d(p_j, r_i) \leq q\ell + (r-1)\ell\}$. Note that S_1 is empty: $d(j,i) > \gamma l_i^{\delta'} \ell^{1-\delta'} = q\ell$ for all $j \in S'$ because i, j are (γ, δ)-independent, and so $S' = \cup_{r=2}^{\infty} S_r$. Let us fix an $r > 1$. Consider any two links $j, k \in S_r$ s.t. $l_j \geq l_k$. We have that $d(p_j, p_k) \geq d(j,k) > \ell$ (1-independence) and that $d(p_j, r_i) \leq \gamma q\ell + (r-1)\ell$ for each $j \in S_r$ (by the definition of S_r), so using the doubling property of the metric space, we get the following bound:

$$|S_r| = |\{p_j\}_{j \in S_r}| \leq C \cdot \left(\frac{q\ell + (r-1)\ell}{\ell}\right)^m = C\,(q + r - 1)^m\,. \tag{2}$$

Note also that $l_j \leq 2\ell$ and $d(i,j) \geq q\ell + (r-2)\ell$ for any link $j \in S_r \setminus S_{r-1}$ with $r > 1$; hence,

$$f_\tau(j,i) = \frac{l_j^{\tau\alpha} l_i^{(1-\tau)\alpha}}{d(i,j)^\alpha} \leq \ell^{(\tau-1)\alpha} l_i^{(1-\tau)\alpha} \left(\frac{2\ell}{q\ell + (r-2)\ell}\right)^\alpha = \frac{2^\alpha \ell^{(\tau-1)\alpha} l_i^{(1-\tau)\alpha}}{(q + r - 2)^\alpha}. \tag{3}$$

Recall that $S_{r-1} \subseteq S_r$ for all $r > 1$, $S_1 = \emptyset$ and $S' = \cup_{r=2}^{\infty} S_r$. Using (3), we have:

$$\begin{aligned} f_\tau(S', i) &= \sum_{r \geq 2} \sum_{j \in S_r \setminus S_{r-1}} f_\tau(j,i) \\ &\leq \sum_{r \geq 2} (|S_r| - |S_{r-1}|) \frac{2^\alpha \ell^{(\tau-1)\alpha} l_i^{(1-\tau)\alpha}}{(q + r - 2)^\alpha} \\ &= 2^\alpha \left(\frac{l_i}{\ell}\right)^{(1-\tau)\alpha} \sum_{r \geq 2} |S_r| \left(\frac{1}{(q + r - 2)^\alpha} - \frac{1}{(q + r - 1)^\alpha}\right), \end{aligned} \tag{4}$$

where the last equality is just a rearrangement of the sum. The sum can be bounded as follows:

$$\begin{aligned} \sum_{r \geq 2} |S_r| \left(\frac{1}{(q + r - 2)^\alpha} - \frac{1}{(q + r - 1)^\alpha}\right) &\leq \sum_{r \geq 2} |S_r| \frac{\alpha}{(q + r - 1)^{\alpha + 1}} \\ &\leq \sum_{r \geq 2} \frac{C\alpha (q + r - 1)^m}{(q + r - 1)^{\alpha + 1}} \\ &= O\left(\sum_{r \geq 2} \frac{1}{(q + r - 1)^{\alpha - m + 1}}\right) \\ &= O\left(\frac{1}{q^{\alpha - m}} + \frac{1}{q^{\alpha - m + 1}}\right) \\ &= O\left(\frac{1}{(\gamma l_i^{\delta'} \ell^{-\delta'})^{\alpha - m}} + \frac{1}{(\gamma l_i^{\delta'} \ell^{-\delta'})^{\alpha - m + 1}}\right) \\ &= O\left(\gamma^{m - \alpha} \left(\frac{\ell}{l_i}\right)^{\delta'(\alpha - m)} \left(1 + \left(\frac{\ell}{l_i}\right)^{\delta'}\right)\right), \end{aligned}$$

where the first line follows from Fact 3, the second one follows from (2) and the fourth one follows from Fact 4. Combined with (4), this completes the proof for the set S'.

The proof holds symmetrically for the set S''. Recall that S'' consists of the links of S which are closer to the sender s_i than to the receiver r_i. Now, we can re-define p_j to denote the endpoint of link j that is closer to s_i, for each $j \in S''$. The rest of the proof will be identical, by replacing r_i with s_i in the formulas. This is justified by the symmetry of (γ, δ)-independence. ◀

In the following two lemmas we bound the affectance of a fixed link i by a set L of independent links that is sufficiently separated from i. The two cases when L consists of links longer than i and shorter than i are treated separately because they impose different conditions on parameters δ and τ. The idea of the proof is to split L into length classes, bound the affectance by each length class using Lemma 5 then combine the obtained bounds in a geometric series.

▶ **Lemma 6.** *Let L be a 1-independent set of links and i be a link s.t. $l_i \geq l_j$ and i, j are (γ, δ)-independent for all $j \in L$. Then for each $\tau > 1 - \delta(1 - m/\alpha)$, $f_\tau(L, i) = O\left(\gamma^{m-\alpha}\right)$.*

Proof. Let us split L into length classes $L_1, L_2, \dots$ with $L_t = \{j \in L : 2^{t-1}\ell \leq l_j < 2^t\ell\}$ where ℓ is the shortest link length in L. Let ℓ_t be the shortest link length in L_t. Note that each L_t is an equilength 1-independent set of links that are (γ, δ)-independent from link i. Thus, the conditions of Lemma 5 hold for each L_t (with $\delta' = \delta$ since all links in L_t are shorter than link i):

$$f_\tau(L_t, i) = O\left(\gamma^{m-\alpha}\left(\frac{\ell_t}{l_i}\right)^{\delta(\alpha-m)-(1-\tau)\alpha}\right).$$

Recall that L_t are equilength sets and $\ell_t \geq 2^{t-1}\ell$. That allows us to combine the bounds above into a geometric series:

$$f_\tau(L, i) = \sum_1^\infty f_\tau(L_t, i) \leq \frac{C \cdot \gamma^{m-\alpha}}{l_i^{(1-\tau)\alpha-\delta(\alpha-m)}} \sum_{t=0}^{\lceil \log l_i/\ell \rceil} (2^t\ell)^{(1-\tau)\alpha-\delta(\alpha-m)},$$

where C is a constant. The upper limit of the last sum is obtained by the fact that link i is not shorter than the longest link in L. Recall that $\tau > 1 - \delta(1 - m/\alpha)$; hence, $\delta(\alpha - m) - (1 - \tau)\alpha > 0$. Thus, the last sum is the sum of a *growing* geometric progression and is $O(l_i^{(1-\tau)\alpha-\delta(\alpha-m)})$, implying the lemma. ◀

▶ **Lemma 7.** *Let L be a 1-independent set of links and i be a link s. t. $l_i \leq l_j$ and i, j are (γ, δ)-independent for all $j \in L$. Then for each $\tau < 1 - (1 - \delta)(\alpha - m + 1)/\alpha$, $f_\tau(L, i) = O\left(\gamma^{m-\alpha}\right)$.*

Proof. Let us split L into length classes $L_1, L_2, \dots$, where $L_t = \{j \in L : 2^{t-1}l_i \leq l_j < 2^t l_i\}$. Note that each L_t is a equilength 1-independent set of links that are (γ, δ)-independent from link i. Let ℓ_t denote the shortest link length in L_t. Recall that $\ell_t \geq 2^{t-1}l_i$. Thus, Lemma 5 implies (note that $\delta' = 1 - \delta$ in this case):

$$f_\tau(L_t, i) = O\left(\gamma^{m-\alpha}\left(\frac{l_i}{\ell_t}\right)^\eta\right) = O\left(\gamma^{m-\alpha}\left(\frac{1}{2^{t-1}}\right)^\eta\right),$$

where $\eta = (1 - \tau)\alpha - (1 - \delta)(\alpha - m + 1)$. Recall that $\tau < 1 - (1 - \delta)(\alpha - m + 1)/\alpha$, implying $\eta > 0$. Thus, we have:

$$f_\tau(L, i) = \sum_1^\infty f_\tau(L_t, i) \leq \gamma^{m-\alpha} \sum_{t=0}^{\lceil \log l_i/\ell \rceil} \frac{1}{2^{\eta t}} = O\left(\gamma^{m-\alpha}\right),$$

where C is a constant. ◀

By combining Lemmas 6 and 7 we find a family of upper bound graphs.

▶ **Corollary 8.** *If $\delta \in (\delta_0, 1)$ and the constant $\gamma > 1$ is large enough, the graphs $\mathcal{G}_\gamma^\delta(L)$ are upper bound graphs for any set L, where $\delta_0 = \frac{\alpha-m+1}{2(\alpha-m)+1}$. Namely, there exists $\tau \in (0,1)$ s.t. any (γ,δ)-independent set is P_τ-feasible. Moreover, we can choose $\tau = \delta$ whenever $\delta > \alpha/(2\alpha - m)$ and $m > 1$.*

Proof. We need to show that Lemmas 6 and 7 hold simultaneously for the given δ and certain $\tau \in (0,1)$. Then we can adjust γ in order to make L feasible. The constraints of the mentioned lemmas on δ and τ are as follows:

$$\tau > 1 - \delta\frac{\alpha - m}{\alpha} \text{ and } \tau < 1 - (1-\delta)\frac{\alpha - m + 1}{\alpha}. \tag{5}$$

So it is enough to show that any $\delta \in (\delta_0, 1)$ is a solution for the following system of inequalities:

$$0 < 1 - \delta\frac{\alpha - m}{\alpha} < 1 - (1-\delta)\frac{\alpha - m + 1}{\alpha} < 1. \tag{6}$$

The first and third inequalities hold whenever $\delta < 1$ and $\alpha > m$. The second inequality is equivalent to $\delta > \delta_0$. The conditions for choosing $\tau = \delta$ follow by setting $\tau = \delta$ in (5). ◀

Putting the Pieces Together. All the components of the conflict graph framework are ready now: a lower bound graph $\mathcal{G}_\gamma$, efficiently colorable upper bound graphs $\mathcal{G}_\gamma^\delta$ and a bound on the gap between the chromatic numbers of those graphs (by Thm. 2). Hence, we can apply the technique described at the beginning of this section to prove the main result – a $O(\log\log\Delta)$-approximation algorithm for Scheduling and WCapacity, using oblivious power schemes.

▶ **Theorem 9.** *There are $O(\log\log\Delta)$-approximation algorithms for Scheduling and WCapacity using oblivious power schemes. The approximation is obtained by approximating vertex coloring or maximum weighted independent set problems in $\mathcal{G}_\gamma^\delta(L)$ with appropriate constants γ and δ.*

4 Limitations of Oblivious Power Schemes

Euclidean Metrics. We proved that it is possible to approximate Scheduling and WCapacity within a factor of $O(\log\log\Delta)$ using oblivious power schemes. As shown in the theorem below, this bound is essentially best possible when using oblivious power assignments. The following was shown in greater generality in [15].

▶ **Theorem 10.** *[15] For every power scheme P_τ, there is an infinite family of feasible sets S arranged in a straight line such that any schedule of S using P_τ requires $\Omega(\log\log\Delta)$ slots.*

Recall that we obtained our approximations only for oblivious power schemes P_τ with τ falling in a specific sub-interval of $(0,1)$. What happens with the other oblivious power schemes? Interestingly, as we show below, oblivious power schemes P_τ with τ outside the range stipulated by Lemmas 6 and 7 yield only $O(\log\Delta)$-approximation for Scheduling and WCapacity.

We consider a family of sets L of $(1,1)$-independent links that are located in the Euclidean plane (hence, $m = 2$). With separation $\delta = 1$, the range of oblivious power schemes P_τ making L (almost) feasible according to Lemmas 6 and 7 is: $2/\alpha < \tau < 1$. In the following theorem

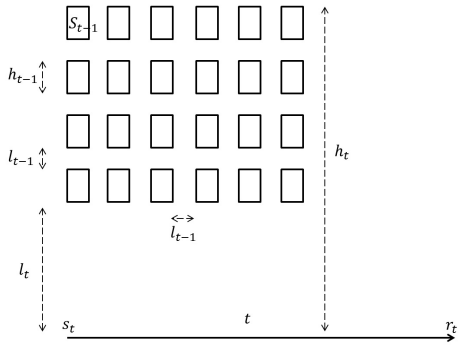


Figure 1 $(1,1)$-independent instance S_t. Each rectangle represents a sub-instance that is a translated copy of S_{t-1}.

we show that no scheduling algorithm can achieve better than $O(\log \Delta)$-approximation of Scheduling for the set L using a scheme P_τ with $\tau < \frac{2}{\alpha}$. An equivalent lower bound applies to WCapacity.

▶ **Theorem 11.** *Any algorithm for Scheduling that uses power assignment P_τ, $\tau < 2/\alpha$, is no better than $\Omega(\log n)$ ($\Omega(\log \Delta)$)-approximate in terms of n (in terms of Δ, resp.). The same holds for WCapacity.*

Proof. We will prove that for infinitely many n, there is a set of n pairwise $(1,1)$-independent links in the plane that requires $\Omega(\log n)$ slots when using P_τ, $\tau < 2/\alpha$. In terms of Δ, the number of slots required is $\Omega(\log \Delta)$.

We assume that $\beta = 1$. We inductively construct a weighted set of links $S_t = S_t(q)$ in the plane, given a parameter q. We shall denote by $S_t^{(x,y)}$ a copy of the instance S_t translated by the vector (x, y).

The instance S_0 consists of the single link 0 of length $l_0 = 1$, with s_0 at the origin and r_0 at $(l_0, 0) = (1, 0)$. For $t \geq 1$, the instance S_t consists of the link t of length $3ql_{t-1}$ and of weight $\omega(t) = q^{2t}$ with s_t at the origin and r_t at $(l_t, 0)$, along with q^2 sub-instances $S_{t-1}^{(x_i,y_j)}$ with $i, j = 0, 1, \ldots q-1$, $x_i = 2il_{t-1}$ and $y_j = l_t + j(l_{t-1} + h_{t-1})$, where h_t is the height of S_{t-1}. This completes the construction. See Figure 1.

It is easily verified that links in S_t are $(1,1)$-independent; hence, S_t can be scheduled in constant number of slots using an oblivious power scheme, by Lemmas 6, 7 and Thm. 1. It remains to show that it requires $\Omega(\log n)$ slots when using P_τ, with $\tau < 2/\alpha$.

Note that the number n_t of links in S_t is $n_t = 1 + q^2 n_{t-1} = \sum_{i=0}^{t-1} q^{2i} = (q^{2t} - 1)/(q^2 - 1)$. Thus, $\log n_t = \theta(t \log q)$. Let us call t the *main link* of S_t. Let us fix an index $t > 0$. Let L_k denote the set of main links of the copies of S_k in S_t, where $k < t$. We call L_k *the k-th level of S_t*. All the links in L_k have equal length and weight q^{2k}. It is easy to check that $W_t = \omega(L_k) = q^{2t}$, and the total weight of all links is $tW_t = \theta(W_t \log n)$.

▶ **Lemma 12.** *Suppose $q \geq (2 \cdot 3^{\tau\alpha})^{1/(2-\tau\alpha)}$. Let T be a subset of links in $S_t(q)$ that is feasible under P_τ with $\tau < 2/\alpha$. Then, $\omega(T) \leq 2 \cdot 3^\alpha W_t$.*

Proof. First, an observation.

▶ **Claim 13.** *Let T_k be a subset of level k links in S_t s.t. $T_k \cup \{t\}$ is P_τ-feasible. Then, $\omega(T_k) \leq 3^\alpha 2^{-(t-k)} W_t$.*

Proof. Let us first estimate the distance $d(i,t)$ for each link $i \in S_t \setminus \{t\}$. Note that $l_t = (3q)^t$, $h_t = l_t + q(l_{t-1} + h_{t-1})$ and $h_0 = 0$ (because S_0 consists of one horizontal link), so we can see that $h_t \le 2l_t$. It follows that for each $i \in S_t \setminus \{t\}$, $d(i,t) \le 3l_t$, implying, for each $i \in T_k$,

$$a_{P_\tau}(i,t) = \frac{P_\tau(i) l_t^\alpha}{P_\tau(t) d_{i,t}^\alpha} = \left(\frac{l_i}{l_t}\right)^{\tau\alpha} \left(\frac{l_t}{d_{it}}\right)^{\alpha} \ge \frac{1}{3^\alpha}(3q)^{-(t-i)\tau\alpha}.$$

Since $a_{P_\tau}(L_k, t) \le 1$, T_k contains at most $3^\alpha(3q)^{(t-i)\tau\alpha}$ links, each of weight q^{2k}, for a total weight of

$$\omega(T_k) \le 3^\alpha(3q)^{(t-k)\tau\alpha} \cdot q^{2k} = 3^\alpha W_t \frac{(3q)^{(t-k)\tau\alpha}}{q^{2(t-k)}} = 3^\alpha W_t \left(\frac{(3q)^{\tau\alpha}}{q^2}\right)^{t-k}.$$

The bound on q ensures that $q^{2-\tau\alpha} \ge 2 \cdot 3^{\tau\alpha}$ or $q^2 \ge 2 \cdot (3q)^{\tau\alpha}$. Thus, $\omega(T_k) \le 3^\alpha W_t (1/2)^{t-k}$, as claimed. ◀

We now prove the lemma by induction on t. For $t = 0$, S_t consists of only one link of weight $1 = q^0 = W_0$. For the inductive step, we consider two cases. Suppose first that T contains the link t. Then, it follows from the claim that

$$\omega(T) \le \omega(t) + \sum_{k=0}^{t-1} \omega(T \cap L_k) \le W_t + 3^\alpha W_t \sum_{k=0}^{t-1} 2^{-(t-k)} < 2 \cdot 3^\alpha W_t.$$

If, on the other hand, T does not contain t, it follows from the inductive hypothesis that the total weight of links from T in each of the q^2 sub-instances $S_{t-1}^{x,y}$ is at most $2 \cdot 3^\alpha W_{t-1}$, for a grand total of $\omega(T) \le q^2 \cdot 2 \cdot 3^\alpha W_{t-1} = 2 \cdot 3^\alpha W_t$. ◀

Observe that the maximum length $\Delta(S_t) = \Delta_t$ of a link in S_t is the length $l_t = (3q)^t$ of link t, which implies that $\log \Delta_t = \theta(t \log q) = \theta(\log n)$. Thus, $\Omega(\log \Delta)$ is also a lower bound. ◀

As for the power schemes P_τ with $\tau \ge 1$, it is known that there is no algorithm using these power schemes that achieves better than $O(\log \Delta)$-approximation in terms of Δ. This is shown in [32] for $\tau = 1$ and easily follows from [38] for $\tau > 1$.

General Metrics. Recall that for Capacity, the $O(\log\log \Delta)$-approximation results hold in arbitrary metrics [18]. This begs the question whether this might also hold for Scheduling and WCapacity. A negative answer was given for Scheduling in [19, Thm. 5.1]: no bound of the form $f(\Delta)$, for any function f of Δ alone. Namely, a feasible instance of n *equal length links* (i.e. $\Delta = 1$) in a tree metric was given in [19], for which P_τ (which is necessarily uniform power (P_0) on equal length links) requires $\Omega(\log n)$ slots. Thus, there is a separation between possible bounds for Capacity and Scheduling. We simplify below this construction and show that it also gives the same lower bound for WCapacity.

▶ **Theorem 14.** *The use of oblivious power assignments cannot obtain approximations of Scheduling or WCapacity within $o(\log n)$ factor in arbitrary general metrics.*

Proof. We give a construction of a set of weighted *equal length* links that is feasible with a certain power assignment, but for which any subset that is feasible using oblivious power, contains at most $\Omega(\log n)$ fraction of the total weight. Since the links have equal lengths, the only possible oblivious assignment is the uniform one. This yields a $\Omega(\log n)$ lower bound on the price of oblivious power for the weighted capacity problem.

The set L of links consists of K subsets, $L_1, L_2, \ldots, L_K$ for $K > 0$. Each set L_k contains 4^{k-1} links, each of weight $1/|L_k|$, for a total weight of 1. L_k also has an associated number $t_k = (\gamma|L_k|)^{\frac{1}{\alpha}}$, for a constant parameter γ to be determined. The distance between a link in L_k and another link in $L_{k'}$ is simply $t_k + t_{k'}$. We assume that $\beta = 1$. This completes the construction. The total number of links $n = |L| = \sum_{k=1\ldots K} |L_k| = (4^K - 1)/3$, and the total weight is K.

It was shown in [19] that L is feasible using some power assignment. We give below a simplified proof. We first show that any feasible set using uniform power has weight $O(1)$, or $O(1/\log n)$-fraction of the whole.

▶ **Claim 15.** *Let $S \subseteq L$ be a subset of links of weight $\omega(S) \geq 1 + \gamma 2^\alpha$. Then, S is infeasible under uniform power.*

Proof. Let $\tilde{k}$ be the minimum value for which an element of $L_{\tilde{k}}$ exists in S. Consider an arbitrary link $l_j \in L_{\tilde{k}} \cap S$. Note that for $i \in L_k$ where $k > \tilde{k}$, $d_{ij} = t_k + t_{\tilde{k}} \leq 2t_k = 2(\gamma|L_k|)^{1/\alpha}$. The affectance $a(i,j) = a_{P_0}(i,j)$ under uniform power is then $a(i,j) = \frac{1}{d_{ij}^\alpha} \geq \frac{1}{\gamma 2^\alpha} \cdot \frac{1}{|L_k|}$. Now,

$$\sum_{i \in S} a(i,j) \geq \sum_{k > \tilde{k}} a(L_k \cap S, j) \geq \frac{1}{\gamma 2^\alpha} \sum_{k > \tilde{k}} \frac{|L_k \cap S|}{|L_k|} = \frac{\omega(S \setminus L_{\tilde{k}})}{\gamma 2^\alpha} \geq \frac{\omega(S) - 1}{\gamma 2^\alpha} > 1 \ . \qquad \blacktriangleleft$$

▶ **Claim 16.** *L is feasible, assuming $\gamma \geq 6$.*

Proof. We will use the power assignment P defined by $P(i) = \frac{1}{2^k}$, for $i \in L_k$. Consider $j \in L_{\tilde{k}}$ and $i \in L_k$, for some $\tilde{k}$, k. Then, $d_{ij}^\alpha > t_{\max(k,\tilde{k})} = \gamma 2^{2(\max(k,\tilde{k})-1)}$. Thus,

$$a_P(L_k, j) = |L_k| \frac{2^{\tilde{k}-k}}{d_{ij}^\alpha} \leq 2^{2(k-1)} \cdot \frac{2^{\tilde{k}-k}}{\gamma \cdot 2^{2(\max(k,\tilde{k})-1)}} = \frac{1}{\gamma} 2^{\min(k,\tilde{k}) - \max(k,\tilde{k}) + 1} \ .$$

It follows that

$$\begin{aligned} a_P(L,j) = \sum_{k > \tilde{k}} a_P(L_k, j) + \sum_{k \leq \tilde{k}} a_P(L_k, j) &< \frac{1}{\gamma} \left(\sum_{k > \tilde{k}} 2^{\tilde{k}-k+1} + \sum_{k \leq \tilde{k}} 2^{k-\tilde{k}+1} \right) \\ &< \frac{1}{\gamma} \left(\sum_{x=0}^{\infty} \frac{1}{2^x} + \sum_{x=0}^{\infty} \frac{2}{2^x} \right) = \frac{6}{\gamma} \ . \end{aligned}$$

Thus, for $\gamma \geq 6$, L is feasible. ◀

Thm. 14 now follows. ◀

5 Weak Links

Recall that in order to obtain our approximations, we assumed that for each link i, $P(i) \geq c\beta N l_i^\alpha$ for a constant $c > 1$. However, this is not always achievable when nodes have limited power. Suppose that each sender node has maximum power P_{max}. For concreteness, we assume that $c = 2$. A link i is called a *weak link* if $P_{max} \leq 2\beta N l_i^\alpha$. Note that a link is weak because it is too long for its maximum power, i.e. if $l_i \geq l_{max}/2^{1/\alpha}$, where $l_{max} = (P_{max}/\beta N)^{1/\alpha}$ is the maximum length a link can have to be able to overcome the noise when using maximum power. Scheduling weak links may be considered as a separate problem. Let τ-WScheduling denote the problem of scheduling weak links with power scheme P_τ. For a weak link i, let us call $e_i = c_i^{1/\alpha} l_i$ the *effective length* of link i and let $\Delta_e(S) = \max_{i,j \in S} e_i/e_j$. One approach to WScheduling is to split the set of weak links

into effective length classes S' with $\Delta_e(S') \leq 2$. Note that in each class, c_i is almost same for all links. Then we can find constant factor approximate scheduling for each of the classes using known algorithms for non-weak scheduling. This leads to a $O(\log \Delta_e)$ approximation. Unfortunately, there is no known algorithm with approximation factor better than $O(\log \Delta_e)$ or $O(\log n)$. The following theorem shows that constant factor approximation of WScheduling is at least as hard as constant factor approximation of Scheduling with fixed uniform power scheme (denoted UScheduling). The proof is omitted due to space limitations.

▶ **Theorem 17.** *There is a polynomial-time reduction from UScheduling to τ-WScheduling for any $\tau \in [0, 1)$, transforming an arbitrary set L of links to a set W of weak links so that the scheduling number of L with P_0 is within a constant factor of the scheduling number of W using P_τ.*

References

1. Chen Avin, Yuval Emek, Erez Kantor, Zvi Lotker, David Peleg, and Liam Roditty. SINR diagrams: Convexity and its applications in wireless networks. *J. ACM*, 59(4), 2012.
2. Marijke Bodlaender and Magnús M. Halldórsson. Beyond geometry: Towards fully realistic wireless models. In *PODC*, 2014.
3. D. Chafekar, V.S. Kumar, M. Marathe, S. Parthasarathy, and A. Srinivasan. Cross-layer latency minimization for wireless networks using SINR constraints. In *Mobihoc*, 2007.
4. Rene L. Cruz and Arvind Santhanam. Optimal Routing, Link Scheduling, and Power Control in Multi-hop Wireless Networks. In *INFOCOM*, 2003.
5. Sebastian Daum, Seth Gilbert, Fabian Kuhn, and Calvin C. Newport. Broadcast in the ad hoc SINR model. In *DISC*, pages 358–372, 2013.
6. Michael Dinitz. Distributed algorithms for approximating wireless network capacity. In *INFOCOM*, pages 1397–1405, 2010.
7. T. ElBatt and A. Ephremides. Joint Scheduling and Power Control for Wireless Ad-hoc Networks. In *INFOCOM*, 2002.
8. A. Fanghänel, T. Kesselheim, H. Räcke, and B. Vöcking. Oblivious interference scheduling. In *PODC*, pages 220–229, August 2009.
9. Alexander Fanghänel, Thomas Kesselheim, and Berthold Vöcking. Improved algorithms for latency minimization in wireless networks. *Theor. Comput. Sci.*, 412(24):2657–2667, 2011.
10. Liqun Fu, Soung Chang Liew, and Jianwei Huang. Power controlled scheduling with consecutive transmission constraints: complexity analysis and algorithm design. In *INFOCOM*, pages 1530–1538. IEEE, 2009.
11. Olga Goussevskaia, Magnús M. Halldórsson, and Roger Wattenhofer. Algorithms for wireless capacity. *IEEE/ACM Trans. Netw.*, 22(3):745–755, 2014.
12. Olga Goussevskaia, Yvonne-Anne Oswald, and Roger Wattenhofer. Complexity in geometric SINR. In *MobiHoc*, pages 100–109, 2007.
13. Helga Gudmundsdottir, Eyjólfur I. Ásgeirsson, Marijke Bodlaender, Joseph T. Foley, Magnús M. Halldórsson, and Ymir Vigfusson. Measurement based interference models for wireless scheduling algorithms. In *MSWiM*, 2014. arXiv:1401.1723.
14. P. Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Trans. Inf. Theory*, 46(2):388–404, 2000.
15. M. M. Halldórsson. Wireless scheduling with power control. *ACM Transactions on Algorithms*, 9(1):7, December 2012.
16. M. M. Halldórsson and P. Mitra. Nearly optimal bounds for distributed wireless scheduling in the SINR model. In *ICALP*, 2011.

17 Magnús M. Halldórsson and Jorgen Bang-Jensen. A note on vertex coloring edge-weighted digraphs. Technical Report 29, Institute Mittag-Leffler, Preprints Graphs, Hypergraphs, and Computing, 2014.

18 Magnús M. Halldórsson, Stephan Holzer, Pradipta Mitra, and Roger Wattenhofer. The power of non-uniform wireless power. In *SODA*, pages 1595–1606, 2013.

19 Magnús M. Halldórsson and Pradipta Mitra. Wireless Capacity with Oblivious Power in General Metrics. In *SODA*, 2011.

20 Magnús M. Halldórsson and Pradipta Mitra. Wireless capacity and admission control in cognitive radio. In *INFOCOM*, pages 855–863, 2012.

21 Magnús M. Halldórsson and Pradipta Mitra. Wireless Connectivity and Capacity. In *SODA*, 2012.

22 Magnús M. Halldórsson and Tigran Tonoyan. How well can graphs represent wireless interference? In *STOC*, 2015.

23 Magnús M. Halldórsson and Roger Wattenhofer. Wireless communication is in APX. In *ICALP*, pages 525–536, 2009.

24 Juha Heinonen. *Lectures on Analysis on Metric Spaces*. Springer, 1 edition, 2000.

25 Tomasz Jurdzinski, Dariusz R. Kowalski, Michal Rozanski, and Grzegorz Stachowiak. On the impact of geometry on ad hoc communication in wireless networks. In *PODC*, pages 357–366, 2014.

26 Bastian Katz, M Volker, and Dorothea Wagner. Energy efficient scheduling with power control for wireless networks. In *WiOpt*, pages 160–169. IEEE, 2010.

27 T. Kesselheim. A Constant-Factor Approximation for Wireless Capacity Maximization with Power Control in the SINR Model. In *SODA*, 2011.

28 T. Kesselheim. Approximation algorithms for wireless link scheduling with flexible data rates. In *ESA*, pages 659–670, 2012.

29 T. Kesselheim and B. Vöcking. Distributed contention resolution in wireless networks. In *DISC*, pages 163–178, August 2010.

30 Henry Lin and Frans Schalekamp. On the complexity of the minimum latency scheduling problem on the Euclidean plane. *arXiv preprint 1203.2725*, 2012.

31 Ritesh Maheshwari, Shweta Jain, and Samir R. Das. A measurement study of interference modeling and scheduling in low-power wireless networks. In *SenSys*, pages 141–154, 2008.

32 Thomas Moscibroda and Roger Wattenhofer. The complexity of connectivity in wireless networks. In *INFOCOM*, pages 1–13, 2006.

33 Thomas Moscibroda, Roger Wattenhofer, and Yves Weber. Protocol design beyond graph-based models. In *HotNets*, 2006.

34 Thomas Moscibroda, Roger Wattenhofer, and Aaron Zollinger. Topology control meets SINR: the scheduling complexity of arbitrary topologies. In *MobiCom*, pages 310–321, 2006.

35 Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 2 edition, 2002.

36 Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental study of concurrent transmission in wireless sensor networks. In *SenSys*, pages 237–250. ACM, 2006.

37 L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. Automat. Contr.*, 37(12):1936–1948, 1992.

38 Tigran Tonoyan. On the capacity of oblivious powers. In *ALGOSENSORS*, pages 225–237, 2011.

39 Tigran Tonoyan. On some bounds on the optimum schedule length in the SINR model. In *ALGOSENSORS*, pages 120–131, 2012.

Allocation of Divisible Goods Under Lexicographic Preferences

Leonard J. Schulman[1] and Vijay V. Vazirani[2]

1 **Caltech, MC305-16, Pasadena CA 91125, USA,** schulman@caltech.edu
2 **College of Computing, Georgia Institute of Technology, Atlanta GA 30332, US,** vazirani@cc.gatech.edu

Abstract

We present a simple and natural non-pricing mechanism for allocating divisible goods among strategic agents having lexicographic preferences. Our mechanism has favorable properties of strategy-proofness (incentive compatibility). In addition (and even when extended to the case of Leontief bundles) it enjoys Pareto efficiency, envy-freeness, and time efficiency.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, J.4 Social and Behavioral Sciences

Keywords and phrases Mechanism design, lexicographic preferences, strategyproof, Pareto optimal, incentive compatible.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.543

1 Introduction

The study of principled ways of allocating divisible goods among agents has long been a central topic in mathematical economics. The method of choice that emerged from this study, the Arrow-Debreu market model [1], provides a powerful approach based on pricing and leads to the fundamental welfare theorems. However, these market-based methods have limitations when agents are assumed to be strategic, e.g., these methods are not incentive compatible. Issues of the latter kind have been studied within the area of mechanism design for the last four decades, and have played a large role in the last decade in algorithmic game theory [20].

In this paper our primary focus is on deriving a non-pricing mechanism for allocating divisible goods, that satisfies incentive-compatibility, Pareto optimality and envy-freeness. A natural approach to achieving Pareto optimality and envy-freeness is to start in a greedy fashion by assigning agents their most favored goods, and gradually moving on to their less favored choices. It is easy to come up with several ways of making this approach precise—two are described in Section 6—and achieve Pareto optimality and envy-freeness. However, it is not a priori clear that it is possible to also achieve incentive compatibility, without which a mechanism is of doubtful merit in an environment of strategic agents. In the main contribution of our paper we show that a third version of this approach, the *Synchronized Greedy (SG) mechanism*, achieves all three properties.

The SG mechanism can be seen as generalizing a mechanism introduced by Crès and Moulin [7], called *Probabilistic Serial (PS)*, in the context of a job scheduling problem, and studied further by Bogomolnaia and Moulin [5] for the allocation of indivisible goods[1]. The

[1] These mechanisms for allocation of indivisible goods are randomized. Our focus on divisible goods is

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 543–559

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

preference model assumed by [5] was first order stochastic dominance, which we will shorten to *sd-preference*. They showed that in this model, PS is efficient, envy-proof and weakly incentive compatible. Furthermore, they showed that in this model, no mechanism satisfies all three properties, i.e., efficiency, envy-proofness and incentive compatibility. In view of the second result, we need to relax the model in order to obtain a mechanism satisfying all three properties; we do so by resorting to the *lexicographic preference relation* and assuming that the goods are divisible.

Lexicographic preferences date back to the work of Hausner [11] and are of interest to economists for the following reasons. They yield a total order on the set of all allocations (unlike sd-preferences, say, which only form a partial order) and they can be seen as a strong-preferences limit of von Neumann-Morgenstern utilities. A preference relation that is complete, transitive and satisfies the continuity condition that preferences between allocations are preserved under limits is known to be representable by a utility function [18]. Of these, lexicographic preferences forgo continuity. What favorable properties can be achieved in the area of goods allocation using only non-pricing mechanisms is a difficult question. The present paper can be regarded as carving out a certain special case, namely the limit in which agents have very strong preferences among the goods, and providing strong positive guarantees in this case. In this limit there is an additional motivation to use non-pricing mechanisms, because very strong preferences might cause a pricing mechanism to do little more than ensure that the wealthiest agents get what they want. By focusing on non-pricing mechanisms, we can study what game-theoretic properties an allocation mechanism can achieve, without depending on what resources the agents possess or care to invest in the game.

There are many every-day examples where something like our model comes up—naturally, not in market economy transactions, but in other societal mechanisms for allocation. An important class is allocation of public resources, e.g., placement lotteries in public schools, see Kojima [16] for further examples and references. (Note also that this kind of example employs a standard reduction of the indivisible goods case to the divisible goods case by randomization.)

The recent paper of Saban and Sethuraman [22] builds on our work and solves several open problems stated in an earlier version of this paper [24]; these results are described at the end of Section 1.2. The broader challenge of the utility-functions version of the allocation problem remains largely open. The simplicity of the SG mechanism is perhaps encouraging toward the existence of allocation mechanisms maintaining favorable (maybe weaker) game-theoretic properties in this setting. Finally, we note that independent of our work, Cho [6] has also studied the use of lexicographic preferences in the context of probabilistically assigning indivisible objects to agents.

Parameters of the problem

In the allocation problem there are m distinct divisible goods which need to be allocated among n agents. Good j ($1 \leq j \leq m$) is available in the amount $q_j > 0$, and agent i ($1 \leq i \leq n$) is to receive a specified $r_i > 0$ combined quantity of all goods; the parameters satisfy $\sum_j q_j \geq \sum_i r_i$, i.e., the total supply is at least as large as the total demand. If this inequality fails, our mechanism may still be run after rescaling expectations so that each

just as general, since an allocation of divisible goods can be used without further modification as a randomized allocation of indivisible goods in the same quantities.

agent i is to receive the quantity $r'_i = r_i(\sum q_j)/(\sum r_\ell)$. So in the sequel we may assume $\sum_j q_j \geq \sum_i r_i$.

Preferences: the non-Leontief case

The *non-Leontief* case of our problem is this. An *allocation* of goods is a list of numbers $a_{ij} \geq 0$, with $\sum_j a_{ij} = r_i$ and $\sum_i a_{ij} \leq q_j$, indicating that agent i receives quantity a_{ij} of good j. The vector $a_{i*} = (a_{i1}, \ldots, a_{im})$ is referred to as agent i's (share of the) allocation. Each agent i has a *preference list*, which is a permutation π_i of the goods; $(a_{i\pi_i(1)}, \ldots, a_{i\pi_i(m)})$ is agent i's *sorted allocation*. Agent i's preference among allocations is induced by *lexicographic order*. That is to say, agent i *lexicographic-prefers* a_{i*} to b_{i*}, written $a_{i*} >_i b_{i*}$, if the leftmost nonzero coordinate of $(a_{i\pi_i(1)}, \ldots, a_{i\pi_i(m)}) - (b_{i\pi_i(1)}, \ldots, b_{i\pi_i(m)})$ is positive. Furthermore, we will say that agent i prefers a_{i*} to b_{i*} in the stochastic domination order [5], or *sd-prefers* a_{i*} to b_{i*}, written $a_{i*} >_i^{\text{sd}} b_{i*}$, if

$$\text{for all } k = 1, \ldots, m: \quad \sum_{\ell=1}^{k} a_{i\pi_i(\ell)} \geq \sum_{\ell=1}^{k} b_{i\pi_i(\ell)},$$

with at least one of the inequalities being strict. The symbols $\geq_i$ and $\not\geq_i$ will have the obvious interpretations.

Since an agent's preferences depend only on his own share of the allocation, we speak interchangeably of an agent's preference for an allocation or an allocation share. In particular, $a_{i*} >_i b_{i*}$ may be written more simply as $a >_i b$, and $a_{i*} >_i^{\text{sd}} b_{i*}$ may be written as $a >_i^{\text{sd}} b$.

Preferences: Leontief Bundles

Some of our results hold in the more general setting of lexicographic preferences among Leontief bundles, and some fail in that setting; details below. A Leontief bundle is specified by a non-negative vector $\lambda = (\lambda_1, \ldots, \lambda_m) \in \mathbb{R}_+^m$ (where $\mathbb{R}_+$ = non-negative reals). The set of goods j for which λ_j is positive is called the *support* of this bundle. (If the set is of size one, we refer to this as a singleton bundle; in Economics this is sometimes also called the linear case.) If $q \in \mathbb{R}_+^m$ then the bundle λ may be allocated from q in any quantity $\alpha \in \mathbb{R}_+$ such that $\alpha\lambda_j \leq q_j$ for all j. In an instance of our problem, a list of M Leontief bundles $\lambda^1, \ldots, \lambda^M$ is specified, including among them the m singleton bundles (hence always $M \geq m$). It is convenient, and in our context sacrifices no generality, to impose the convention that for every bundle λ^k, $\sum_1^m \lambda_j^k = 1$.

The case $m = M$, in which all bundles are singletons, is of course a special case of the Leontief framework, but to distinguish it from the general situation we call it the "non-Leontief" case.

The framework we are concerned with is that each agent i has a preference list specified by a permutation π_i of the bundles. A Leontief allocation is an $n \times M$ matrix ℓ in which ℓ_{ik} represents the quantity of bundle k allocated to agent i. A Leontief allocation l imposes the goods allocation $A(l)$, an $n \times m$ matrix, by $A(l)_{ij} = \sum_{k=1}^{M} l_{ik}\lambda_j^k$. We further require that a Leontief allocation satisfy the conditions $\sum_j A(l)_{ij} = r_i$ (thanks to the convention above this is equivalent to $\sum_k l_{ik} = r_i$) and $\sum_i A(l)_{ij} \leq q_j$. We speak of $A(l)_{i*}$ and l_{i*} as agent i's *share* of, respectively, the goods and the Leontief bundles. The vector $(l_{i\pi_i(1)}, \ldots, l_{i\pi_i(M)})$ is agent i's *sorted Leontief share*. Agent i's preference among allocations is induced by *lexicographic order* on his share of the allocation. That is to say, agent i *lexicographic-prefers* l to l', written $l >_i l'$, if the leftmost nonzero coordinate of $(l_{i\pi_i(1)}, \ldots, l_{i\pi_i(M)}) - (l'_{i\pi_i(1)}, \ldots, l'_{i\pi_i(M)})$ is positive. Thus, for any goods allocation a, there is a favored Leontief allocation, denoted

$L^\pi(a)$, defined by providing each agent with the best Leontief share that can be assembled from his share of the goods—to be explicit, this is obtained by starting with a_{i*} as the available goods vector, and then, for k from 1 to M, setting $L^\pi(a)_{i\pi_i(k)}$ to be the largest α such that ((available goods vector)$-\alpha\lambda^k) \in \mathbb{R}^M_+$, then subtracting $\alpha\lambda^k$ from the available goods vector and iterating.

We say that agent i *sd-prefers* allocation a to b, written $a >_i^{\mathrm{sd}} b$, if

$$\text{for all } K = 1, \ldots, M: \quad \sum_{k=1}^{K} L^\pi(a)_{i\pi_i(k)} \geq \sum_{k=1}^{K} L^\pi(b)_{i\pi_i(k)},$$

with at least one of the inequalities being strict.

The two orders

Observe that "lexicographic-prefers" is a complete preference relation without indifference contours (since it is antisymmetric for distinct allocation shares), and that "sd-prefers" is an incomplete preference relation; moreover the lexicographic order is a refinement of the sd order, i.e., sd-prefers implies lexicographic-prefers. The phrase "agent i weakly X-prefers" will be used to include the possibility that agent i's share is identical in the two allocations.

1.1 Our results

The SG mechanism is deterministic, treats all agents symmetrically, and has the following properties.

Properties w.r.t. sd preference

- If all r_i's are equal, the allocation produced by the SG mechanism in response to truthful bids is envy-free in the following sense: each agent weakly sd-prefers his allocation to that of any other agent. This holds also in the Leontief case.

Properties w.r.t. lexicographic preference

(Since most of our paper deals with the relation "lexicographic-prefers", we subsequently abbreviate it to "prefers".)

- The allocation produced by the SG mechanism in response to truthful bids is Pareto efficient. This holds also in the Leontief case.
- Incentive compatibility for a single agent: In the non-Leontief case, the SG mechanism is strategy-proof if $\min_j q_j \geq \max_i r_i$.
  We give counterexamples (a) in the absence of this inequality, (b) for the Leontief case.
- Generalizing the previous item, we have: Incentive compatibility for a coalition: The SG mechanism is group strategy-proof against coalitions of ℓ agents if
  $$\min_j q_j \geq \max_{S:|S|=\ell} \sum_{i \in S} r_i$$
  .
- The running time to implement the SG mechanism is $\tilde{O}(mn)$ in the non-Leontief case, and $\tilde{O}(n(m^2 + M))$ in the Leontief case.
- *Any* Pareto efficient allocation can be produced using a suitable "variable speeds" extension of the SG mechanism. This holds also in the Leontief case. (However, the variable speeds extension does not possess the rest of the properties listed above.)

  The incentive compatibility properties are the main results of this paper.

1.2 Literature

There has been considerable work on the strategy-proof allocation of divisible goods in Arrow-Debreu economies, starting with the seminal work of Hurwicz [12], e.g., see [8, 14, 23, 25, 26, 28]. Most of these results are negative, among the recent ones being Zhou's result showing that in a 2-agent, n-good pure exchange economy, there can be no allocation mechanism that is efficient, non-dictatorial (i.e., both agents must receive non-zero allocations) and strategy-proof [28].

The paper that is most closely related to our work is that of Bogomolnaia and Moulin [5]. In their setting there are n agents and n indivisible goods, each agent having a total preference ordering over the goods; the desired outcome is a matching of goods with agents. A straightforward mechanism for allocating one good to each agent is *random priority (RP):* pick a uniformly random permutation of the agents and ask each agent in turn to select a good among those left. It is easy to see that this mechanism is *ex post efficient*, i.e., the allocation it produces can be represented as a probability distribution over Pareto efficient deterministic allocations, and it is strategy-proof. However, it is not ex ante efficient. A random allocation is said to *ex ante efficient* if for any profile of von Neumann-Morgenstern utilities that are consistent with the preferences of agents, the expected utility vector is Pareto efficient. It is easy to see that ex ante efficiency implies ex post efficiency.

Solving a conjecture of Gale [9], Zhou [27] showed that no strategy-proof mechanism that elicits von Neumann-Morgenstern utilities and achieves Pareto efficiency can find a "fair" solution even in the weak sense of equal treatment of equals. He further showed that the solution found by RP may not be efficient if agents are endowed with utilities that are consistent with their preferences. Hence, ex ante efficiency had to be sacrificed, if strategy-proofness and fairness were desired.

In the face of these choices, the work of Bogomolnaia and Moulin gave the notion of *ordinal efficiency* that is intermediate between ex post and ex ante efficiency; an allocation a is ordinally efficient if there is no other allocation b such that every agent sd-prefers b to a. They went on to show that the mechanism called *probabilistic serial (PS)*, introduced in Crès and Moulin [7], yields an ordinally efficient allocation. Further they show that PS is envy-free and weakly strategy-proof, defined appropriately for the partial order "sd-prefers". Finally, Bogomolnaia and Moulin define an extension of PS by introducing different "eating rates" and show that this set of mechanisms characterizes the set of all ordinally efficient allocations.

Katta and Sethuraman [15] generalize the setting of Bogomolnaia and Moulin to the "full domain", i.e., agents may be indifferent between pairs of goods. Thus, each agent partitions the goods by equality and defines a total order on the equivalence classes of her partition (the agent is equally happy with any good received from an equivalence class). For this setting, they give a randomized mechanism that is a generalization (different from ours) of PS and achieves the same game-theoretic properties as PS.

A mechanism that probabilistically allocates indivisible goods can also be viewed as one that fractionally allocates divisible goods. Under the latter interpretation, the SG mechanism is equivalent to PS for the case that $m = n$ and the quantity of each good and the requirement of each agent is one unit. An important difference is that Bogomolnaia and Moulin analyze PS under an incomplete preference relation (stochastic dominance) in which "most" allocation shares are incomparable; whereas we analyze SG under a complete preference relation (lexicographic) that is a refinement of stochastic dominance. The statement that a mechanism's allocation is Pareto efficient w.r.t. lexicographic preferences is considerably stronger than the same statement w.r.t. stochastic dominance preferences, because each

agent's share is dominated by more alternative shares in the lexicographic order, than it is in the sd order; so, fewer allocations are Pareto efficient in the lexicographic than in the sd order. Our results should be viewed therefore as demonstrating that the PS mechanism and its natural generalization, SG, have far stronger game-theoretic properties than even envisioned in [5].

For somewhat related questions primarily regarding exchange economies, see Barberà and Jackson [4], Nicolo [19], Ghodsi et al. [10], and Li and Xue [17]. Finally, we remark only that the problem of allocating a *single* divisible good among multiple agents with known privileges is considerably different; the principal issue studied in that problem is how to make the division in a manner that is fair w.r.t. the given privileges. This is known as the bankruptcy problem and has a long history, e.g., see [21, 2]. Despite an interesting resemblance between the PS mechanism and some of the mechanisms used in the solutions of that problem [13], the issues at stake in the bankruptcy literature are distinct from those in our paper and its predecessors.

Saban and Sethuraman [22] solve some of the open problems stated in an earlier version of this paper. They consider the special case that all $r_i = 1$. First they show that our condition $\min_j q_j \geq \max_i r_i$ is tight in the sense that for any $q_1 < 1$ there exists an n, a finite list $q_2, \ldots, q_n$, and agent preferences such that no mechanism is efficient, envy-free and strategyproof. They also show that if $q_1 < 1$, and list $q_2, \ldots, q_n$ and the agent preferences are given, then SG achieves all three properties if and only if any mechanism achieves all three properties. Finally for the generalized setting of Katta and Sethuraman, where agents can be indifferent between objects, they show that no mechanism can satisfy all three properties.

Since the PS rule is not strategyproof, recent work has studied the situation where agents are strategic. A Nash equilibrium for the PS rule is a preference profile for which no agent has an incentive to report a different profile. [3] show that a pure Nash equilibrium is guaranteed to exist; however determining whether a given preference profile is a Nash equilibrium is coNP-complete.

2 The Synchronized Greedy Mechanism

The mechanism is simple. Each agent i submits a preference list σ_i. The submitted list may or may not, of course, agree with his true preference list π_i.

(A simple case to consider is that of $M = m = n$ and all $q_j = r_i = 1$. Because of the restriction that each preference list must include all m singleton bundles, each agent's preference list in this case is a permutation of the m goods. Despite being quite special, this case, or the slightly more general case in which $M = m \leq n$ and all r_i are equal, is already interesting to analyze and is well motivated by the examples, mentioned earlier, involving sharing of tasks or of scarce public resources.)

The mechanism simulates the following physical process. Consider each good j as a "liquid", and each agent as a receptacle of capacity r_i. The mechanism starts out at time 0 by (for all i in parallel) pouring bundle $\lambda^{\sigma_i(1)}$ into receptacle i at rate r_i units of liquid per unit time. Each good j is therefore being drained at rate $\sum_i r_i \lambda_j^{\sigma_i(1)}$. (Note that since $\sum_j \lambda_j = 1$, the total liquid being added to receptacle i per unit time is r_i, as desired.)

This continues until one of the goods, say j, is exhausted. For all agents who were currently being allocated bundles with j in their support, their favorite Leontief bundle has now been exhausted. (We say that a Leontief bundle has been *exhausted* at a given time if any of the goods in its support has been exhausted, and otherwise that the bundle is *available*.) All such agents, i, are immediately allocated the next available bundle on their preference list,

and the pouring of bundles continues. The algorithm continues in this way, allocating to an agent from the next available bundle whenever the current bundle has been exhausted. Since the singleton bundles are included in all preference lists, all agents continuously receive goods at rate r_i until time 1, at which time they simultaneously complete their full allocation.

Observe that the Leontief allocation l constructed by SG satisfies $l = L^{\pi}(A(l))$ because the bundles are provided to each agent greedily based on the availability of goods.

This continuous process can easily be converted into a discrete algorithm with the run time cited earlier: maintain a priority queue of goods, keyed by termination times. Each time a good is exhausted, each agent is assigned its next unexhausted bundle, and an updated termination time for each good is computed using the coefficients of the active bundles.

Observe that if an agent prefers bundle λ to bundle λ', and $\text{support}(\lambda) \subseteq \text{support}(\lambda')$, then λ' may be removed from the agent's preference list. It cannot be allocated to the agent by SG nor can it be part of any Pareto efficient allocation to the agent.

3 Properties of the Synchronized Greedy Mechanism

3.1 Pareto Efficiency

Let l^{σ} be the allocation created by the SG mechanism in response to bids σ declared by the agents. As before π denotes the truthful bids.

▶ **Theorem 1.** *The allocation produced by the SG mechanism in response to truthful bids is Pareto efficient w.r.t. lexicographic preference. That is to say, for all $l \neq l^{\pi}$, $\exists i$ $l <_i l^{\pi}$.*

Proof. For agent i and for $K \geq 1$ let $t_{iK} = \frac{1}{r_i} \sum_{k=1}^{K} l^{\pi}_{i\pi_i(k)}$. If agent i receives a positive quantity of his K'th-most-favored bundle, then t_{iK} is the time when that bundle is exhausted in SG. If the agent receives nothing from the bundle then the bundle is exhausted in SG no later than t_{iK}.

Suppose for contradiction the existence of l s.t. $\forall i$ $l \geq_i l^{\pi}$, and for some i, $l >_i l^{\pi}$. Let t be minimum s.t. $\exists i, K$ s.t. $t = t_{iK} < \frac{1}{r_i} \sum_{k=1}^{K} l_{i\pi_i(k)}$. Note, if $t_{i'K'} < t$ then $t_{i'K'} = \frac{1}{r_{i'}} \sum_{k=1}^{K'} l_{i'\pi'_i(k)}$.

For every one of the bundles $b \in \{\pi_i(1), \ldots, \pi_i(K)\}$ there is a good $j(b)$ that appears positively in b and which is exhausted by time t. Since $t_{iK} < \frac{1}{r_i} \sum_{k=1}^{K} l_{i\pi_i(k)}$ while $t_{iK'} = \frac{1}{r_i} \sum_{k=1}^{K'} l_{i\pi_i(k)}$ for all $K' < K$, some agent $i' \neq i$ receives strictly less of good $j(\pi_i(K))$ in l than in l^{π}. Since $j(\pi_i(K))$ is exhausted in SG by time t, this means that there is some K'' such that $\frac{1}{r_{i'}} \sum_{k=1}^{K''} l_{i'\pi_{i'}(k)} < \frac{1}{r_{i'}} \sum_{k=1}^{K''} l^{\pi}_{i'\pi_{i'}(k)} \leq t$. This contradicts the minimality of t. ◀

3.2 Strategy-Proofness

A mechanism is said to be *strategy-proof* if for every agent and for every list of bids by the remaining agents, the agent cannot obtain a strictly improved allocation by lying.

▶ **Theorem 2.** *In the non-Leontief case, the SG mechanism is strategy-proof if* $\min q_j \geq \max r_i$.

Proof. Without loss of generality focus on agent 1. For the remainder of this proof $\pi_2, \ldots, \pi_n$ are arbitrary bids by the agents $2, \ldots, n$, but π_1 is agent 1's truthful bid. We need to show that for any bid σ_1 (and write $\sigma = (\sigma_1, \pi_2, \ldots, \pi_n)$), $a^{\sigma}_{1*} \leq_1 a^{\pi}_{1*}$. The theorem is trivial if $a^{\sigma} = a^{\pi}$.

The theorem is also trivial if agent 1, bidding truthfully, receives only his top choice. So we may suppose that agent 1 does not receive the entire allocation of any one good.

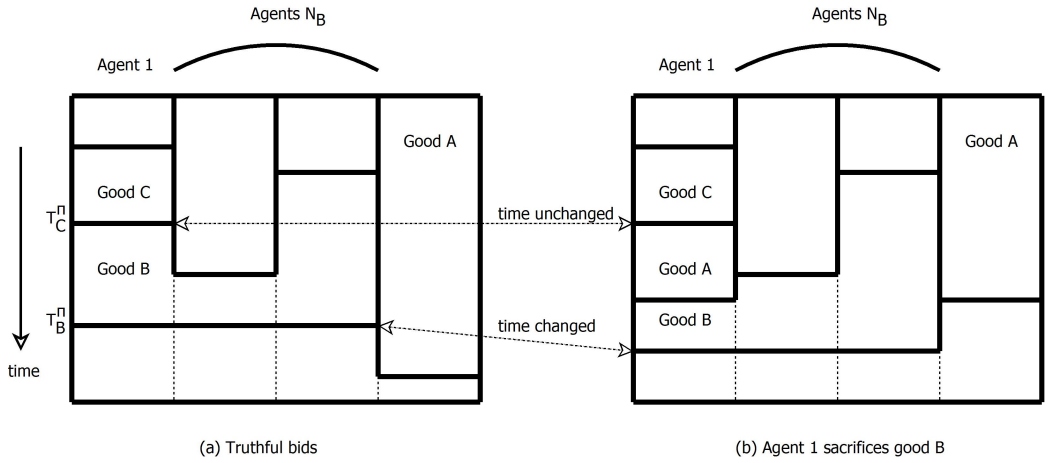


Figure 1 The mechanism with truthful vs. lying bids of Agent 1.

We may also suppose that if $a^{\sigma}_{1j} = 0$ and $a^{\sigma}_{1j'} > 0$, then $\sigma_1^{-1}(j) > \sigma_1^{-1}(j')$. (Define $\sigma_1^{-1}(j)$ to be the s such that $\sigma_1(s) = j$. Define $\pi_1^{-1}(j)$ analogously.) In other words, all the requests in σ_1 that come up empty may as well be deferred to the end.

Let $G(j) = \{j' : \pi_1^{-1}(j') \leq \pi_1^{-1}(j) \text{ and } a^{\pi}_{1j'} > 0\}$. These are the goods that agent 1 weakly prefers to good j and receives a positive quantity of in the allocation a^{π}.

Say that agent 1 *sacrifices* good j in σ if:

1. $a^{\pi}_{1j} > 0$,
2. $\sigma_1^{-1}(j) > |G(j)|$, and
3. $\pi_1^{-1}(j) < \pi_1^{-1}(j')$ if j' also satisfies (1),(2).

That is to say, j is the most-preferred good which agent 1 receives a positive quantity of in π, but requests later in σ than in π.

For a collection of bids ρ let T^{ρ}_j be the time at which good j is exhausted if the mechanism is run with bids ρ.

Agent 1 must sacrifice some good, call it B, since otherwise the allocation will not change. See Figure 1. We will show that agent 1 receives strictly less of B in σ than in π, and that this is not compensated for by getting more of more-preferred goods.

▶ **Lemma 3.** *If D is a good and $T^{\pi}_D < T^{\pi}_B$, then $T^{\sigma}_D \leq T^{\pi}_D$.*

Proof. Supposing the contrary, let D be a counterexample minimizing T^{π}_D. Since $T^{\pi}_D < T^{\pi}_B$, $D \neq B$. Now let i be any agent (who may or may not be agent 1) for whom $a^{\pi}_{iD} > 0$. Due to the minimality of D, each of the goods j which i prefers in π to D, has $T^{\sigma}_j \leq T^{\pi}_j$. Therefore i requests D at a time in σ that is at least as soon as the time i requests it in π.

Since this holds for all i who received a positive allocation of D in π, the lemma follows. ◀

Let N_B be the set of agents $i \neq 1$ for whom $a^{\pi}_{iB} > 0$. The condition on r_i's and q_j's ensures that this set is nonempty.

Due to the lemma, for each agent in N_B, the request time for B in σ is weakly earlier than it is in π. Now let C be the good such that $\pi_1^{-1}(C)$ is maximal subject to $\pi_1^{-1}(C) < \pi_1^{-1}(B)$ and $a^{\pi}_{1C} > 0$. Due to the lemma, all goods j' such that $\pi_1^{-1}(j') \leq \pi_1^{-1}(C)$ have $T^{\sigma}_{j'} \leq T^{\pi}_{j'}$. Next we show:

▶ **Proposition 4.** *If $\pi_1^{-1}(j') \leq \pi_1^{-1}(C)$, then $a^{\sigma}_{1j'} = a^{\pi}_{1j'}$.*

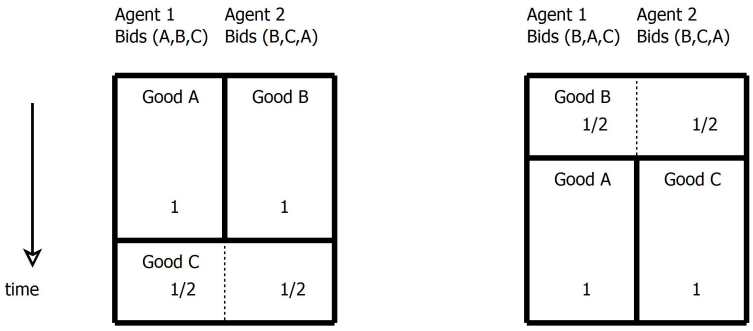


Figure 2 Failure of strategy-proofness without the hypothesis of Theorem 2.

Proof. Supposing the contrary, let $\pi_1^{-1}(j')$ be minimal such that $\pi_1^{-1}(j') \leq \pi_1^{-1}(C)$ and $a^{\sigma}_{1j'} \neq a^{\pi}_{1j'}$. There are two possibilities to consider.
(a) $a^{\sigma}_{1j'} < a^{\pi}_{1j'}$. This is not possible because then $a^{\sigma}_{1*} <_1 a^{\pi}_{1*}$.
(b) $a^{\sigma}_{1j'} > a^{\pi}_{1j'}$. Note:

▶ **Lemma 5.** *Let j_1, j_2 be such that $\pi_1^{-1}(j_1) \leq \pi_1^{-1}(B)$, $\pi_1^{-1}(j_2) \leq \pi_1^{-1}(B)$, $a^{\pi}_{1j_1} > 0$, and $\pi_1^{-1}(j_1) < \pi_1^{-1}(j_2)$. Then $\sigma_1^{-1}(j_1) < \sigma_1^{-1}(j_2)$.*

Proof. Consider the least j_1 that is part of a pair j_1, j_2 violating the lemma. Then j_1 satisfies conditions (1),(2) above, contradicting that B is the good sacrificed by agent 1. ◀

It follows that $T^{\sigma}_{j'} \geq \sum_{j'':\pi_1^{-1}(j'')\leq\pi_1^{-1}(j')} a^{\sigma}_{1j''}$. Due to the minimality of j', this means that if $a^{\sigma}_{1j'} > a^{\pi}_{1j'}$, then $T^{\sigma}_{j'} > T^{\pi}_{j'}$, contradicting our earlier conclusion. This completes demonstration of the Proposition. ◀

A consequence of the Proposition is that $T^{\sigma}_C = T^{\pi}_C$.

Since agent 1 sacrifices B, his request time for B in σ is strictly greater than his request time for B in π.

Recall that N_B is nonempty. At time T^{π}_B, the agents of N_B have received as least as much of B in σ as they have in π, and the latter is positive. On the other hand, at the same time T^{π}_B, agent 1 has received strictly less of B in σ than he has in π. In order for agent 1 to receive at least as much of B in σ as in π, he would have to receive all of B that is allocated after time T^{π}_B; however, that is not possible, because the set of agents receiving B after T^{π}_B includes N_B. Thus $a^{\sigma}_{1*} <_1 a^{\pi}_{1*}$. ◀

3.3 Necessity of a Hypothesis on $\{r_i\}, \{q_j\}$s

We next provide an example in which strategy-proofness fails in the absence of the condition $\max r_i \leq \min q_j$. For convenience now let $r_1 \geq \ldots \geq r_n$ and $q_1 \leq \ldots \leq q_m$.

▶ **Example 6.** Let $n = 2$ and $m = 3$. Let $r_1 = r_2 = 3/2$; label the goods A, B, C, let $q_A = q_B = q_C = 1$, and let the preference lists be $\pi_1 = (A, B, C)$, $\pi_2 = (B, C, A)$. If agent 1 bids truthfully he receives the sorted allocation $(1, 0, 1/2)$. If instead he bids (B, A, C) (while agent 2 bids truthfully), he receives the improved sorted allocation $(1, 1/2, 0)$. See Figure 2.

This example does not limit the theorem sharply, because it uses $r_1 = (3/2)q_1$ rather than r_1 arbitrarily close to q_1. Jeremy Hurwitz has pointed out that one may construct similar examples whenever $r_1 \geq q_1/(1 - q_2/\sum q_j)$; this would appear to be a tight bound.

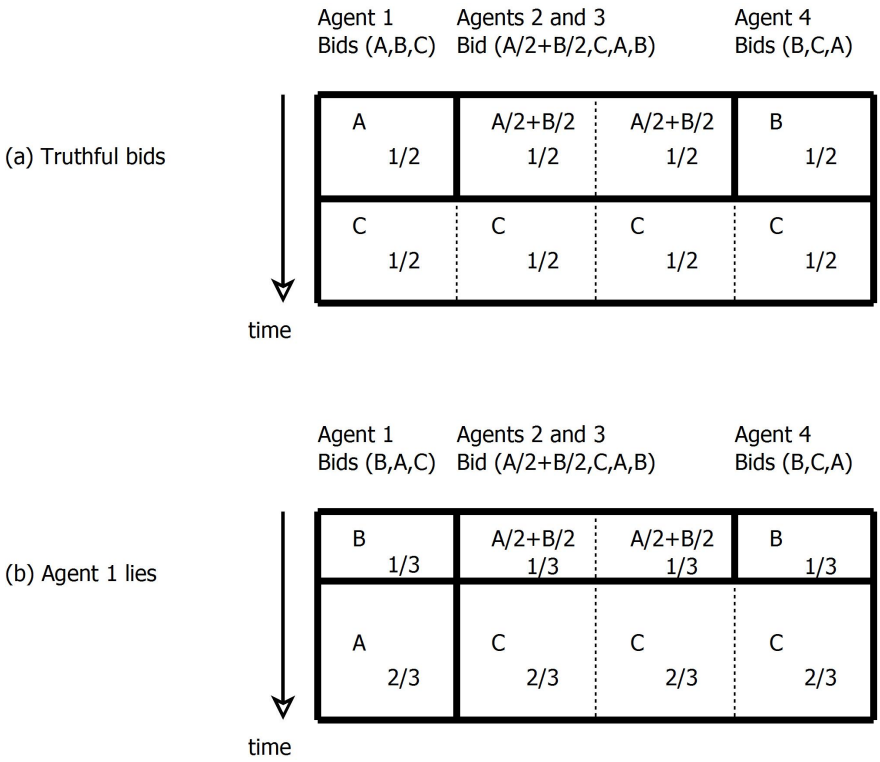


Figure 3 Failure of strategy-proofness in the Leontief case.

3.4 Failure of strategy-proofness for the Leontief case

Theorem 2 has no equivalent for general Leontief bundles. Consider the following four-agent system with $r_1 = r_2 = r_3 = r_4 = 1$ and three goods in supply $q_A = q_B = 1, q_C = 2$. Agent 1's desired Leontief bundles are in the preference order (A, B, C) (this agent is interested only in singleton bundles); agent 2 and 3's desired Leontief bundles are in the order $(\frac{1}{2}A + \frac{1}{2}B, C, A, B)$; agent 4's Leontief bundles are in the order (B, C, A).

Under truthful bidding agent 1 receives the sorted goods allocation $(1/2, 0, 1/2)$. By bidding instead (B, A, C), agent 1 receives the improved sorted goods allocation $(2/3, 1/3, 0)$. See Figure 3.

3.5 Group Strategy-Proofness

A mechanism is *group strategy-proof* against a family F of subsets of agents if for every "coalition" $S \in F$ and for any list of bids by the agents outside of S, the agents of S cannot obtain an improved allocation by lying, where by "improved allocation" we mean that no agent of S obtains a worse allocation and at least one obtains a strictly better allocation.

We now provide the following generalization of Theorem 2:

▶ **Theorem 7.** *In the non-Leontief case, the SG mechanism is group strategy-proof against the family of subsets S for which* $\min_j q_j \geq \sum_{i \in S} r_i$.

▶ **Corollary 8.** *In the non-Leontief case, the SG mechanism is group strategy-proof against coalitions of ℓ agents if* $\min_j q_j \geq \max_{S:|S|=\ell} \sum_{i \in S} r_i$.

The proof of Theorem 7 follows a structure similar to that of Theorem 2 but the argument is complicated by the fact that different agents in S can sacrifice different goods, and some of the agents may actually be better off due to their untruthful bids (as they may benefit from the interactions among the several lies). The proof needs to effectively "chase through" an unbounded iteration of good transfers relative to a^π, and show that *some* agent in the coalition is worse off than in π. Fortunately, this can be done without explicitly pursuing the iteration.

Proof. Let S be a minimal counterexample. That is,

(a) $\min_j q_j \geq \sum_{i \in S} r_i$;

(b) With π_i representing in this proof the truthful preferences for $i \in S$ and arbitrary preferences for $i \notin S$, there are bids σ_i for $i \in S$ such that every $i \in S$ "is a willing participant in the coalition S", namely (with $\sigma_\ell = \pi_\ell$ for $\ell \notin S$) $a^\sigma_{i*} \geq_i a^\pi_{i*}$;

(c) For some $i \in S$, $a^\sigma_{i*} >_i a^\pi_{i*}$;

(d) No strict subset of S satisfies (a),(b),(c).

Note by minimality that in σ, every agent $i \in S$ bids untruthfully (differently from π) and this has an effect, namely, if i reverts to bidding according to π then the allocation is different than in σ.

If $a^\pi_{i\pi_i(1)} = r_i$ for all $i \in S$, that is, with truthful bids these agents receive only their top choices, then none of them can be strictly rewarded by submitting a different bid.

Otherwise (i.e., if $a^\pi_{i\pi_i(1)} < r_i$ for some $i \in S$), then thanks to the hypothesis, under the truthful bids π, every good has a positive allocation outside S.

We may simplify the argument slightly by supposing that for each agent $i \in S$, if $a^\sigma_{ij} = 0$ and $a^\sigma_{ij'} > 0$, then $\sigma_i^{-1}(j) > \sigma_i^{-1}(j')$. In other words, all the requests that come up empty may as well be deferred to the end.

Let $G(i,j) = \{j' : \pi_i^{-1}(j') \leq \pi_i^{-1}(j) \text{ and } a^\pi_{ij'} > 0\}$.

Say that agent i *sacrifices* good j in σ if:

1. $a^\pi_{ij} > 0$,
2. $\sigma_i^{-1}(j) > |G(i,j)|$, and
3. $\pi_i^{-1}(j) < \pi_i^{-1}(j')$ if j' also satisfies (1),(2).

Some good must be sacrificed by some agent, since otherwise the allocation will not change. (However, while every agent in S is untruthful, not every $i \in S$ necessarily sacrifices a good; setting $\sigma_i(j) > \pi_i(j)$ might have an effect even if $a^\pi_{ij} = 0$ because of increased availability of j due to bidding changes of other agents.)

Of all the sacrificed goods let B be one for which T^π_B is minimal.

▶ **Lemma 9.** *If D is a good and $T^\pi_D < T^\pi_B$, then $T^\sigma_D \leq T^\pi_D$.*

Proof. Supposing the contrary, let D be a counterexample minimizing T^π_D. By the minimality of B, D cannot be a sacrificed good.

Now let i be any agent (inside or outside of S) for whom $a^\pi_{iD} > 0$. Due to the minimality of D, each of the goods j which i truthfully prefers to D, has $T^\sigma_j \leq T^\pi_j$. Therefore i requests D at a time in σ that is at least as soon as the time i requests it in π.

Since this holds for all i who received a positive allocation of D in π, the lemma follows. ◀

Let $O_B \subseteq S$ be the set of agents who sacrifice B, and let N_B be the set of agents i for whom $a^\pi_{iB} > 0$ but who do not sacrifice B. Due to the lemma, for each agent in N_B, the request time for B in σ is weakly earlier than it is in π. Now consider an agent $i \in O_B$. Let C be the good such that $\pi_i^{-1}(C)$ is maximal subject to $\pi_i^{-1}(C) < \pi_i^{-1}(B)$ and $a^\pi_{iC} > 0$. Due to the lemma, all goods j' such that $\pi_i^{-1}(j') \leq \pi_i^{-1}(C)$ have $T^\sigma_{j'} \leq T^\pi_{j'}$. Next we show:

▶ **Proposition 10.** *If* $\pi_i^{-1}(j') \leq \pi_i^{-1}(C)$, *then* $a^{\sigma}_{ij'} = a^{\pi}_{ij'}$.

Proof. Supposing the contrary, let $\pi_i^{-1}(j')$ be minimal such that $\pi_i^{-1}(j') \leq \pi_i^{-1}(C)$ and $a^{\sigma}_{ij'} \neq a^{\pi}_{ij'}$. There are two possibilities to consider.
(a) $a^{\sigma}_{ij'} < a^{\pi}_{ij'}$. This is not possible because i is a willing participant in the coalition.
(b) $a^{\sigma}_{ij'} > a^{\pi}_{ij'}$. Note:

▶ **Lemma 11.** *Let* j_1, j_2 *be such that* $\pi_i^{-1}(j_1) \leq \pi_i^{-1}(B)$, $\pi_i^{-1}(j_2) \leq \pi_i^{-1}(B)$, $a^{\pi}_{ij_1} > 0$, *and* $\pi_i^{-1}(j_1) < \pi_i^{-1}(j_2)$. *Then* $\sigma_i^{-1}(j_1) < \sigma_i^{-1}(j_2)$.

Proof. Identical to the proof of Lemma 5 with agent i in place of agent 1. ◀

It follows that $T^{\sigma}_{j'} \geq \sum_{j'':\pi_i^{-1}(j'')\leq\pi_i^{-1}(j')} a^{\sigma}_{ij''}$. Due to the minimality of j', this means that if $a^{\sigma}_{ij'} > a^{\pi}_{ij'}$, then $T^{\sigma}_{j'} > T^{\pi}_{j'}$, contradicting our earlier conclusion. This completes demonstration of the Proposition. ◀

A consequence of the Proposition is that $T^{\sigma}_{C} = T^{\pi}_{C}$.

Since agent i sacrifices B, his request time for B in σ is strictly greater than his request time for B in π.

Since we are in the case that every good has a positive allocation outside S, N_B is nonempty. At time T^{π}_{B}, the agents of N_B have received as least as much of B in σ as they have in π, and the latter is positive. On the other hand, at the same time T^{π}_{B}, the agents of O_B have received strictly less of B in σ than they have in π. In order for the agents of O_B to receive collectively at least as much of B in σ as in π, they would have to receive all of B that is allocated after time T^{π}_{B}; however, that is not possible, because the set of agents receiving B after T^{π}_{B} includes N_B. Therefore there is some $i \in O_B$ for whom $a^{\sigma}_{iB} < a^{\pi}_{iB}$. This contradicts the requirement that i be a willing participant in the coalition S. ◀

▶ **Example 12.** Example 6, in which strategy-proofness failed absent the hypothesis of Theorem 2, can be extended in a straightforward manner to one in which the group strategy-proof property fails to hold absent the hypothesis of Corollary 8. Again use $m = 3$, but instead of two agents, use $n = 2\ell$ agents, the first half having the same preference order (A, B, C) as agent 1 in the earlier example, and the second half having the same preference order (B, C, A) as agent 2 in the earlier example. If all agents bid truthfully, then the first ℓ agents each receive the sorted allocation $(1, 0, 1/2)$; however if they lie and bid (B, A, C), while the remainder bid truthfully, then each lying agent receives the improved sorted allocation $(1, 1/2, 0)$.

4 Characterizing All Pareto Efficient Allocations

Bogomolnaia and Moulin [5] extended their mechanism by allowing players to receive goods at time-varying rates. Specifically, for each agent i there is a speed function η_i mapping the time interval $[0, 1]$ into the nonnegative reals, such that for all i, $\int_0^1 \eta_i(t)\, dt = r_i$. Subject to these speeds, goods flow to agents in order of the preference lists they bid, just as before. They showed that this extension characterizes all ordinally efficient allocations.

In this section, we obtain an analogous characterization of all Pareto efficient allocations by a similar extension of our mechanism. Specifically, we prove that for *any* Pareto efficient allocation of bundles, there exist speeds such that the extended SG mechanism produces that allocation. We prove this after first noting that the extended SG mechanism always results in Pareto efficient allocations.

In this section when η_i $(1 \le i \le n)$ are fixed, we let a^π (with the η's implicit) be the goods allocation produced by the extended SG mechanism with these speeds and truthful bids. We let $l^\pi = L^\pi(a^\pi)$ be the corresponding allocation of bundles.

4.1 Pareto Efficiency

▶ **Theorem 13.** *Let η_i, $1 \le i \le n$, be any speed functions. Then the allocation l^π is Pareto efficient.*

Proof. The argument is the same as for Theorem 1 with the proviso that the definition $t_{iK} = \frac{1}{r_i}\sum_{k=1}^{K} l^\pi_{i\pi_i(k)}$ is replaced by $t_{iK} = \inf\{y : \int_0^y \eta_i(t)\,dt \ge \sum_{k=1}^{K} l^\pi_{i\pi_i(k)}\}$. ◀

4.2 Characterizing All Pareto Efficient Allocations

If the last result mirrored the First Welfare Theorem, the next mirrors the Second Welfare Theorem:

▶ **Theorem 14.** *Let π be the collection of agent preference lists over bundles, and let l be a Pareto efficient allocation. There exist speed functions η_i, $1 \le i \le n$, such that $l = l^\pi$.*

Proof. As before the bundles are $(\lambda^k)_{k=1}^M$, where for each k, $\sum_{j=1}^m \lambda_j^k = 1$, and $\lambda_j^k \ge 0$ for all j.

Construction of the speeds η_i is simple. Let a "partial bundle allocation" be a list $\hat{l}_{ik}$, each $\hat{l}_{ik} \ge 0$, such that for every i, $\sum_{k,j} \hat{l}_{ik}\lambda_j^k \le r_i$, and for every j, $\sum_{i,k} \hat{l}_{ik}\lambda_j^k \le q_j$.

Initialize $t = 0$ and initialize each agent i with the empty partial allocation $\hat{l}_{ik} = 0$ for all i, k.

Initialize c_j to be the quantity of good j that is allocated in l. (Necessarily $c_j \le q_j$ and $\sum c_j = \sum r_i$. If $\sum q_j > \sum r_i$ then for some j, $c_j < q_j$.)

Then repeat the following until $t = 1$.

Find an agent i for whom there is an ℓ such that $\hat{l}_{i\pi_i(\ell)} < l_{i\pi_i(\ell)}$, and such that for all $\ell' < \ell$, the bundle $\pi_i(\ell')$ has been exhausted (that is to say, there is a good j such that $\lambda_j^{\pi_i(\ell')} > 0$ and $c_j = 0$.) To see that there is such an i, suppose the contrary, and consider all the agents for whom $\sum_{k,j} \hat{l}_{ik}\lambda_j^k < r_i$. For each of them there is a favorite bundle which has not yet been exhausted. Evidently none of these agents is to be allocated in l any additional quantity of this favorite bundle. However since these favorite bundles have not yet been exhausted, we can allocate to every player a slight additional positive amount of his favorite unexhausted bundle, without exhausting any additional goods. Any extension of this new partial bundle allocation to a full bundle allocation, strictly Pareto dominates l, contrary to assumption.

Now set $\delta = (l_{i\pi_i(\ell)} - \hat{l}_{i\pi_i(\ell)})/\sum r_i$. For $t < t' < t + \delta$, make the settings $\eta_i(t') = \sum r_i$ and, for $i' \ne i$, $\eta_{i'}(t') = 0$. Then increment $\hat{l}_{i\pi_i(\ell)}$ by $\delta \sum r_i$, and decrement each c_j by the corresponding amount, namely, decrement c_j by $\lambda_j^{\pi_i(\ell)}\delta\sum r_i$. Finally, increment t by δ.

This process terminates in finitely many iterations because in each iteration some agent completes its allocation of some bundle. ◀

Examination of the above proof reveals:

▶ **Corollary 15.** *There is a polynomial time algorithm for checking whether a given allocation is Pareto efficient.*

4.3 No Incentive Compatibility for the Variable Speeds Variant

We note that the synchrony imposed among agents by the SG mechanism is key to its incentive compatibility and envy-freeness properties (indeed, the properties hold even if the basic mechanism is extended with the *same* speed function for all agents). If different agents have different speed functions under the extended SG mechanism, Theorems 2 and 7, showing incentive compatibility, fail to hold. The argument breaks down as soon as it uses termination times, in Lemma 3. Below is a counter-example for strategy-proofness; a similar idea gives counter-examples for group strategy-proofness and envy-freeness.

▶ **Example 16.** Assume $m = n = 4$ and that all $r_i = q_j = 1$. Let the speed function for agent 1 be 1 over the interval $[0, 1]$. The speeds of agents 2, 3, and 4 equal 1 over the interval $[0, 1/2]$, 0 over the interval $(1/2, 5/6]$, and 3 over the interval $(5/6, 1]$. The preference orders of agents 1 and 2 are $(1, 2, 3, 4)$, and the preference orders of agents 3 and 4 are $(2, 4, 3, 1)$. If all agents bid truthfully, agent 1 receives the sorted allocation $(1/2, 0, 1/2, 0)$. On the other hand, if agent 1 bids $(2, 1, 3, 4)$ while the rest bid truthfully, then agent 1 receives the better sorted allocation $(1/2, 1/3, 1/6, 0)$.

5 Envy-Freeness w.r.t. stochastic dominance preference

(This section is the only part of the paper where we use sd preference.)

Given a bundle allocation l, let $\bar{l}$ denote the *relative allocation,* where $\bar{l}_{ij} = l_{ij}/r_i$.

▶ **Theorem 17.** *Under truthful bidding, every agent i weakly sd-prefers his relative allocation $\bar{l}^{\pi}_{i*}$ to the relative allocation $\bar{l}^{\pi}_{i'*}$ of any other agent i'.*

Proof. Fix any $1 \leq k \leq M$. We are to show that

$$\frac{1}{r_i}\sum_{\ell=1}^{k} l^{\pi}_{i\pi_i(\ell)} \geq \frac{1}{r_{i'}}\sum_{\ell=1}^{k} l^{\pi}_{i'\pi_i(\ell)}.$$

Let t be the time at which the last of the bundles $\pi_i(1), \ldots, \pi_i(k)$ is exhausted. So $tr_i = \sum_{\ell=1}^{k} l^{\pi}_{i\pi_i(\ell)}$. No other agent can receive any of these bundles after time t, so $tr_{i'} \geq \sum_{\ell=1}^{k} l^{\pi}_{i'\pi_i(\ell)}$. ◀

6 Other Greedy Mechanisms

As stated in the Introduction, obtaining an efficient and envy-free non-pricing mechanism for allocating divisible goods is easy, but additionally satisfying incentive compatibility is harder. In this section we present two greedy mechanisms which satisfy the first two properties but not the third. To simplify description of the mechanisms, assume that $m = n$ and that all $r_i = q_j = 1$; it is straightforward to generalize the mechanisms beyond this restriction, and our counterexamples are possible even with it.

Mechanism 1: The mechanism proceeds iteratively. In round i, it considers the ith-favorite goods of all agents who still have not been allocated a full unit of goods. Among such agents, if the ith-favorite good of a set S of agents is good j, the remaining quantity of good j is allocated equally among the agents in S, subject to no agent getting more than a total of one unit of goods. (Some of good j may remain after the round.)

Mechanism 2: The mechanism has a notion of time, similar to SG. Goods allocation starts at time 0 and is completed at time 1. During this interval each agent receives goods at rate 1. The interval is punctuated by finitely many critical instants at which some of the agents switch which good they are receiving. The first critical instant is 0 and the others are the times at which some nonempty set of agents T finishes receiving their promised allocation of a good. At such an instant, the mechanism identifies, for each of the agents in T, the next-favorite good on their list that has not yet been fully promised to other agents. The mechanism promises each agent in T some of that good, in the following fashion: let T_j be the subset of T requesting good j and let u be the amount of good j that has not been previously promised. Then each agent in T_j is promised an equal share of u subject to no agent exceeding a total of one unit of goods. (The next critical instant affecting these agents is of course easily computed.) The mechanism then proceeds to the next critical instant.

The proofs given above, for showing that the SG mechanism is efficient and envy-free, extend easily to showing that Mechanisms 1 and 2 are also efficient and envy-free. Here, however, are counterexamples to incentive compatibility:

▶ **Example 18** (Mechanism 1)**.** Let $m = n = 4$; name the goods $A, \ldots, D$. Agent 1's preference list is A, B, C, D; agents 2 and 3 have preferences A, C, B, D; and agent 4's favorite good is B. If the agents bid truthfully then in round 1, agent 4 is allocated all of good B, while the first three agents are each allocated a third of good A. In the second round agent 1 is left out while agents 2 and 3 are allocated half of good C. In round 3 no allocations are made, and in round 4 good D is allocated among the first three agents. The allocation to agent 1 is therefore $(A : 1/3, D : 2/3)$. If instead agent 1 submits the preference list A, C, B, D then she is treated the same as agents 2 and 3, and her allocation is $(A : 1/3, C : 1/3, D : 1/3)$, which she prefers.

The counterexample for the second mechanism is more involved.

▶ **Example 19** (Mechanism 2)**.** Let $m = n = 8$; name the goods $A, \ldots, H$. We specify only the essential components of the preference orders. The preference order of agent 1 is alphabetical, $(A, \ldots, H)$. Agents $2, 3, 4$ have the preference order $(A, G, H, F, \ldots)$. Agents $5, 6, 7$ have the preference order $(B, C, E, F, \ldots)$. Agent 8 has the preference order $(B, D, \ldots)$. If all agents report their preferences truthfully, agent 1 gets the allocation $(A : 1/4, C : 1/4, D : 1/4, F : 1/4)$; if agent 1 lies and reports the order $(A, C, E, D, \ldots)$ she gets the allocation $(A : 1/4, C : 1/4, D : 1/4, E : 1/4)$, which she prefers.

7 Discussion

Our main open problem is the one mentioned in the Introduction, i.e., achieving approximate versions of the properties of the SG mechanism but when agents' preferences are representable by utility functions.

Another natural open question concerns the existence of mechanisms to produce lexicographically most equitable allocations, having favorable algorithmic and game-theoretic properties (esp., incentive compatibility). The SG mechanism is not very equitable: see the full paper [24].

Acknowledgments. We are indebted to Hervé Moulin for generously sharing his deep understanding of this research domain. Thanks also to Jeremy Hurwitz for stimulating discussions and to Amin Saberi for pointing us to useful references.

Schulman was supported in part by NSF Grants 1038578 and 1319745. Part of this work was done while visiting the Simons Institute for the Theory of Computing at UC Berkeley.

Vazirani was supported in part by NSF Grants CCF-0914732 and CCF-1216019, and a Google Research Grant. He would like to thank the Social and Informational Sciences Laboratory at Caltech for their hospitality; part of this work was done while he was Distinguished SISL Visitor during 2011-12. He would also like to thank the Guggenheim Foundation for a Fellowship during 2011-12.

References

1 K. Arrow and G. Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22:265–290, 1954.

2 R. Aumann and M. Maschler. Game theoretic analysis of a bankruptcy problem from the Talmud. *J. Economic Theory*, 36:195–213, 1985.

3 H. Aziz, S. Gaspers, S. Mackenzie, N. Mattei, N. Narodytska, and T. Walsh. Equilibria under the probabilistic serial rule. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI'15, pages 1105–1112, 2015.

4 S. Barberà and M. O. Jackson. Strategy-proof exchange. *Econometrica*, 63(1):51–87, 1995.

5 A. Bogomolnaia and H. Moulin. A new solution to the random assignment problem. *Journal of Economic Theory*, 100:295–328, 2001.

6 W. J. Cho. Probabilistic assignment: A two-fold axiomatic approach. 2012.

7 H. Crès and H. Moulin. Scheduling with opting out: improving upon random priority. *Operations Research*, 49(4):565–577, 2001.

8 P. Dasgupta, P. Hammond, and E. Maskin. The implementation of social choice rules. *Review of Economic Studies*, 46:153–170, 1979.

9 D. Gale. College course assignments and optimal lotteries. Mimeo, University of California, Berkeley, 1987.

10 A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *Proc. 8'th USENIX conference on networked systems design and implementation*, 2011.

11 M. Hausner. Multidimensional utilities. In R. L. Davis, R. M. Thrall, and C. H. Coombs, editors, *Decision Processes*, pages 167–180. Wiley, 1954.

12 L. Hurwicz. On informationally decentralized systems. In C. B. McGuire and R. Radner, editors, *Decision and Organization*, pages 297–336. North-Holland, Amsterdam, 1972.

13 M. Kaminski. Hydraulic rationing. *Mathematical Social Sciences*, 40:131–155, 2000.

14 M. Kato and S. Ohseto. Toward general impossibility theorems in pure exchange economies. *Social Choice and Welfare*, 19:659–664, 2002.

15 A. Katta and J. Sethuraman. A solution to the random assignment problem on the full preference domain. *Journal of Economic Theory*, 131:231–250, 2006.

16 F. Kojima. Random assignment of multiple indivisible objects. *Mathematical Social Sciences*, 57(1):134–142, 2009.

17 J. Li and J. Xue. Egalitarian division under Leontief preferences. Manuscript, 2012.

18 A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic theory*. Oxford University Press, 1995.

19 A. Nicolo. Efficiency and truthfulness with Leontief preferences. *Review of Economic Design*, 8(4):373–382, 2004.

20 N. Nisan. Introduction to mechanism design (for computer scientists). In N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, pages 209–241. Cambridge University Press, 2007.

21 B. O'Neill. A problem of rights arbitration from the Talmud. *Mathematical Social Sciences*, 2(4):345–371, 1982.

22 D. Saban and J. Sethuraman. A note on object allocation under lexicographic preferences. *J. Mathematical Economics*, 50:283–289, 2014.

23 M. Satterthwaite and H. Sonnenschein. Strategy-proof allocation mechanisms at differentiable points. *Review of Economic Studies*, 48:587–597, 1981.

24 L.J. Schulman and V. V. Vazirani. Allocation of divisible goods under lexicographic preferences. In arXiv, 1206.4366, 2012.

25 S. Serizawa. Inefficiency of strategy-proof rules for pure exchange economies. *Journal of Economic Theory*, 106:219–241, 2002.

26 S. Serizawa and J. Weymark. Efficient strategy-proof exchange and minimum consumption guarantees. *Journal of Economic Theory*, 109:246–263, 2003.

27 L. Zhou. On a conjecture by Gale about one-sided matching problems. *J. Econ. Theory*, 52:123–135, 1990.

28 L. Zhou. Inefficiency of strategy-proof allocation mechanisms in pure exchange economies. *Social Choice and Welfare*, 8(3):247–254, 1991.

On the Expressiveness of Multiparty Sessions

Romain Demangeon[1] and Nobuko Yoshida[2]

1 **Sorbonne Université, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005 Paris, France**
romain.demangeon@lip6.fr

2 **Imperial College, London, United Kingdom**
yoshida@doc.ic.ac.uk

Abstract

This paper explores expressiveness of asynchronous multiparty sessions. We model the behaviours of endpoint implementations in several ways: (i) by the existence of different buffers and queues used to store messages exchanged asynchronously, (ii) by the ability for an endpoint to lightly reconfigure his behaviour at runtime (flexibility), (iii) by the presence of explicit parallelism or interruptions (exceptional actions) in endpoint behaviour. For a given protocol we define several denotations, based on traces of events, corresponding to the different implementations and compare them.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases concurrency, message-passing, session, asynchrony, expressiveness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.560

1 Introduction

Asynchronous Multiparty Sessions. In large-scale distributed infrastructures, most interactions are based upon the production of interleaving flows of messages between independent participants. Verification of such distributed protocols is challenging: participants are executing applications written in different languages and the way messages are treated between production and consumption may vary. The presence of intermediate layers where on-transit messages are stored and transferred via, *e.g.* buffers or queues, makes the analyses difficult, even with the guarantee that the order of messages is preserved for each intermediate structure.

The approach of *multiparty session types* [11, 7] (extended from the binary [10]) introduced a flexible formal method for verification of message-passing protocols without central control: the desired interactions at the scale of the network itself are specified into a session (called *global type*). These formal objects describe interactions between all participants through simple syntax including send and receive operations, choice and recursion. Global types are then projected onto several *local types* (one for each participant), which describe the protocol from a local point of view. These local types are used to validate an application through type-checking or monitoring. Theory of session types guarantees that local conformance of all participants induces global conformance of the network to the initial global type. Sessions type theory is well-studied and gave birth to languages such as *Scribble* [22], directly inspired by formal session types, letting developers specify and verify (through automatically generated monitors) distributed protocols and applications, *e.g.* for large cyberinfrastructures [8] and business protocols [15].

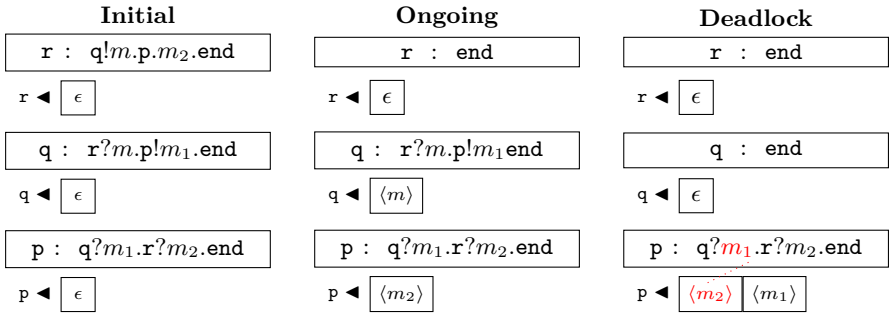


Figure 1 Configurations with single input queues for G.

Although various extensions of multiparty sessions [11] are studied, several fundamental open problems remain, such as *expressiveness* questions: whether *permutations* of types (used to compensate the order of arrival of messages from different sources) [16, 6, 17] and *interruptible* sessions [8] are more expressive than standard sessions or not. We require a canonical methodology to compare these extensions systematically.

Session type expressiveness. This paper explores and compares expressiveness of different semantics for asynchronous multiparty sessions in the literature, based on message traces.

We first study the effect of buffers – order-preserving stores for in-transit messages – on the expressiveness. For instance, adding buffers on the sender side (messages are stored in a queue after being produced and before being transferred to the receiver) is innocuous, whereas receiver-side buffers can produce deadlocks. As an example, consider global type $G = \mathtt{r} \to \mathtt{q} : m, \mathtt{q} \to \mathtt{p} : m_1, \mathtt{r} \to \mathtt{p} : m_2.\mathtt{end}$ which consists of a sequence of three messages exchanged between three participants. It is projected to local types $\mathtt{p} : \mathtt{q}?m_1.\mathtt{r}?m_2.\mathtt{end}$, $\mathtt{q} : \mathtt{r}?m.\mathtt{p}!m_1.\mathtt{end}$ and $\mathtt{r} : \mathtt{q}!m.\mathtt{p}!m_2.\mathtt{end}$, in which each participant $(\mathtt{p}, \mathtt{q}, \mathtt{r})$ is expected to perform two consecutive actions. $\mathtt{q}!m$ is the output of message m to $\mathtt{q}$ and $\mathtt{r}?m$ is the input of message m from $\mathtt{r}$. $\mathtt{end}$ denotes termination. If each participant uses one buffer on the receiver-side (called *input queue*), message m_2 can be arrived and be enqueued in the structure of $\mathtt{p}$ before m_1, leading to a deadlock (as $\mathtt{p}$ expects to consume m_1 first), as described in Figure 1. There exist several ways to allow usage of buffers on the receiving side without risking deadlocks. First, one can separate the input queue into several input queues (as in [7]), one for each possible sender, a program being allowed to consume messages from any queues. In our example, m_2 (coming from $\mathtt{r}$) and m_1 (coming from $\mathtt{q}$) would be stored in different input queues at $\mathtt{p}$, allowing m_1 not to be blocked by an early arrival of m_2. This situation is described in Figure 2. Alternatively, one can introduce flexibility in the program running at $\mathtt{p}$ to make it able to accept m_2 before m_1. Formally, it boils down to a permutation of $\mathtt{p} : \mathtt{q}?m_1.\mathtt{r}?m_2.\mathtt{end}$ to $\mathtt{p} : \mathtt{r}?m_2.\mathtt{q}?m_1.\mathtt{end}$, adapting it to the order of arrival of m_1 and m_2.

Configurations and traces. The common framework we use to describe session networks is *configurations* (drawn from [2]), which are collections of local types – the remaining expected actions for all participants – and queues – the order-preserving structures storing in-transit messages. For instance, the deadlocked situation explained above is described by configuration $\mathtt{p} : \mathtt{q}?m_1.\mathtt{r}?m_2.\mathtt{end}, \mathtt{q} : \mathtt{end}, \mathtt{r} : \mathtt{end}, (\mathtt{p} \blacktriangleleft : \langle \mathtt{q}, \mathtt{p}, m_1 \rangle.\langle \mathtt{r}, \mathtt{p}, m_2 \rangle)$ where $\mathtt{q}$ and $\mathtt{r}$ are finished, $\mathtt{p}$ expects to first receive m_1 then m_2 and the (only) input queue at $\mathtt{p}$ is ready to deliver m_2 then m_1.

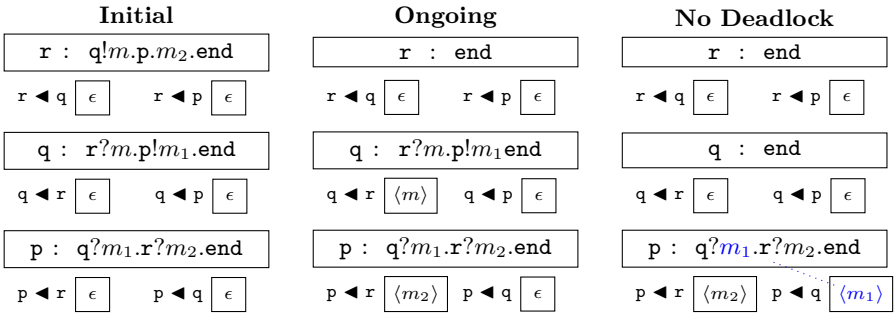


Figure 2 Configurations with multiple input queues for G.

From these configurations, we extract *traces* to compare semantics. They are mappings from participants to sequences of actions, send or receive *events*, ordered *locally*: two events at the same location are ordered, but two events performed by different participants are not. As an example, a trace σ leading to the configuration above from the initial configuration is s.t. $\sigma(\mathtt{p}) = \epsilon$, $\sigma(\mathtt{q}) = \mathtt{r}?m.\mathtt{p}!m_1$, $\sigma(\mathtt{r}) = \mathtt{q}!m.\mathtt{p}!m_2$, describing the fact that m_1 and m_2 have been sent by $\mathtt{r}$ and $\mathtt{q}$, but not yet received by $\mathtt{p}$. The traces contains no information on whether m_1 was sent before of after m_2.

A protocol can then be given a *denotation*, w.r.t. a given semantics, as a set of completed local traces, that is, traces of configurations which cannot progress further. Different semantics yield different denotations for the same type; for instance, the denotation of G under a semantics with simple input queues contains traces stopped at deadlocked configurations (such as σ), whereas the denotation of G under a semantics with multiple input queues (as described above) will only contain completed traces (traces reaching a configurations where all local types are $\mathtt{end}$).

Parallel and interruptible sessions. Next we study the impact on expressiveness of two different constructs: the parallel composition, explicitly notifying that two actions can appear in any order and interruptions. Interruptible sessions have been studied in [12, 8] through the use of *scopes* describing sessions in which a participant can, at any time, raise a interruption to stop the current block of interactions. Suppose $\{\!|\mathtt{p} \to \mathtt{q} : m_1.\mathtt{q} \to \mathtt{r} : m_2.\mathtt{end}|\!\}^{\mathtt{c}}\langle i \text{ by } \mathtt{p}\rangle; \mathtt{p} \to \mathtt{r} : m_3$. In $\{\!|G|\!\}$, m_2 is supposed to be sent by $\mathtt{q}$ after receiving m_1. Participant $\mathtt{p}$ can interrupt the session at any time, as specified in $\langle i$ by $\mathtt{p}\rangle$, for instance after sending m_1, by broadcasting the message i. If i reaches $\mathtt{q}$ after m_1 is received and before m_2 is sent, $\mathtt{q}$ will not send m_2 and the session continues with message m_3 from $\mathtt{p}$ to $\mathtt{r}$.

Contributions. This paper systematically compares the expressiveness of different semantics of multiparty session types based on: (*i*) the presence and the nature of different data structures used to store messages on either side of communications, (*ii*) the flexibility of the local types – defined as a subtyping relation, and (*iii*) the presence of parallel and interruptions.

For the first time, we use sets of languages of local traces to compare expressiveness. We prove, for (*i*) that the introduction of universal input queues (buffer storing incoming messages regardless of their provenance) leads to deadlock but that in absence of such structure, the denotation of any session G stays the same, regardless of the structures used.

We then introduce *flexible subtyping* (*ii*) which permutes the order of local actions in a limited way. We explain how the combination of flexibility and queues can lead to deadlocks and prove that using flexibility yields greater expressive power. Finally (*iii*), we claim that session parallelism and interruption have greater expressiveness using our local trace formalism.

2 Multiparty Session Types

Sessions, seen as protocol specifications, are described by *global types* G [11, 7], the main objects being compared in this work. A global type specifies the interactions expected to happen in a session, between several participants (denoted by $\mathtt{p}, \mathtt{q}, \mathtt{r}$), seen from an omniscient point of view. Syntax of the global and local types is given by:

$$\begin{array}{rcl} G & ::= & \mathtt{end} \mid \mu \mathtt{t}.G \mid \mathtt{t} \mid \mathtt{r}_1 \to \mathtt{r}_2\{m_i.G_i\}_{i\in I} \\ T & ::= & \mathtt{end} \mid \mu \mathtt{t}.T \mid \mathtt{t} \mid \mathtt{p}?\{m_i.T_i\}_{i\in I} \mid \mathtt{p}!\{m_i.T_i\}_{i\in I} \end{array}$$

We call the different $(m_i)_{i\in I}$ *sets of messages*. Type $\mathtt{end}$ is a termination of session, which we sometimes omit. $\mu\mathtt{t}.G$ and $\mathtt{t}$ are the recursion operators. We manipulate equirecursive types, not distinguishing between $\mu\mathtt{t}.G$ and $G[\mu\mathtt{t}.G/\mathtt{t}]$. We assume recursion variables $\mathtt{t}$ are guarded, i.e. they appear only under some prefix. $\mathtt{r}_1 \to \mathtt{r}_2\{m_i.G_i\}_{i\in I}$ is the basic interaction inside global types: participant $\mathtt{r}_1$ is expected to send message m_j to participant $\mathtt{r}_2$ – we assume $\mathtt{r}_1 \neq \mathtt{r}_2$; according to the j chosen by the sender, the protocol will continue as global type G_j. We write $\mathtt{p} \to \mathtt{q} : m_1.G_1$ when $|I| = 1$. We sometimes write $\mathtt{q}?$ or $\mathtt{r}!$ when the message is not relevant.

Local types describe these protocols from the point of view of a participant and are considered as local guidelines distributed processes must follow. Interactions are decomposed into two sides: input $\mathtt{p}?\{m_i.T_i\}_{i\in I}$ and output $\mathtt{p}!\{m_i.T_i\}_{i\in I}$. Local types are effectively (potentially infinite) trees of input and output actions. We often write $\mathtt{p}!m.T$ or $\mathtt{p}?m.T$ for a singleton and $\mathtt{p}!$ if the message is not important.

Projections. Local types are obtained from global types through projection $G\upharpoonright(\mathtt{r})$ (the projection of a global type G onto a participant $\mathtt{r}$). Projection is given by the following rules:

$$\begin{array}{rcl} \mu\mathtt{t}.G\upharpoonright(\mathtt{r}) = \mu\mathtt{t}.G\upharpoonright(\mathtt{r}) \quad (\text{if } G\upharpoonright(\mathtt{r}) \neq \mathtt{t}) & & \mathtt{end}\upharpoonright(\mathtt{r}) = \mathtt{end} \qquad \mathtt{t}\upharpoonright(\mathtt{r}) = \mathtt{t} \\ & & \mu\mathtt{t}.G\upharpoonright(\mathtt{r}) = \mathtt{end} \qquad (\text{otherw.}) \\ \mathtt{r}_1 \to \mathtt{r}_2\{m_i.G_i\}_{i\in I}\upharpoonright(\mathtt{r}) & = & \mathtt{r}!\{m_i.G_i\upharpoonright(\mathtt{r})\}_{i\in I} \quad (\text{if } \mathtt{r} = \mathtt{r}_1) \\ \mathtt{r}_1 \to \mathtt{r}_2\{m_i.G_i\}_{i\in I}\upharpoonright(\mathtt{r}) & = & \mathtt{r}?\{m_i.G_i\upharpoonright(\mathtt{r})\}_{i\in I} \quad (\text{if } \mathtt{r} = \mathtt{r}_2) \\ \mathtt{r}_1 \to \mathtt{r}_2\{m_i.G_i\}_{i\in I}\upharpoonright(\mathtt{r}) & = & G_1\upharpoonright(\mathtt{r}) \quad (\text{otherw., and } \forall i,j \in I.G_i\upharpoonright(\mathtt{r}) = G_j\upharpoonright(\mathtt{r})) \end{array}$$

Recursive global types are projected into recursive local types except when projection name $\mathtt{r}$ does not appear in a recursion block, i.e. $\mathtt{r}$ is not involved in the recursion, thus projection is $\mathtt{end}$. When projecting an communication, if the projection name is the sender (resp. the receiver), the result will be a send (resp. receive) action. If the name is not involved in the communication, the first branch is chosen to continue projection. In the last rule, a choice made during a communication is unobservable to other participants, hence projections in all branches are the same (see [11]). We call projectable global types *well-formed* and assume all types are well-formed in the following.

▶ **Example 1** (Projection). Consider $G = \mathtt{p} \to \mathtt{q} : m.\mathtt{q} \to \mathtt{p} : m_1.\mathtt{r} \to \mathtt{p} : m_2.\mathtt{end}$, described above. This global type describes a session composed of three interactions: $\mathtt{r}$ sends a message m to $\mathtt{q}$ which then sends a message m_1 to $\mathtt{p}$ and finally $\mathtt{r}$ sends a message m_2 to $\mathtt{p}$. Projection of G onto its three participants gives: $\{\mathtt{r} : \mathtt{q}!m.\mathtt{p}!m_2, \quad \mathtt{q} : \mathtt{r}?m.\mathtt{p}!m_1, \quad \mathtt{p} : \mathtt{q}?m_1.\mathtt{r}?m_2\}$.

$$
\begin{array}{lrcll}
\text{(Com)} & \mathtt{p} : \mathtt{q}!\{m_i.T_i\}_{i\in I}, \mathtt{q} : \mathtt{p}?\{m_i.T_i\}_{i\in I} & \xrightarrow{\mathtt{pq}:m_j} & \mathtt{p} : T_j, \mathtt{q} : T_j & j \in I \\
\text{(InIn)} & \mathtt{q} : \mathtt{p}?\{m_i.T_i\}_{i\in I}, (\mathtt{q} \triangleleft \mathtt{p} : \langle \mathtt{p}, \mathtt{q}, m_j\rangle.h) & \xrightarrow{\mathtt{p?q}:m_j} & \mathtt{p} : T_j, (\mathtt{q} \triangleleft \mathtt{p} : h) & j \in I \\
\text{(OutIn)} & \mathtt{p} : \mathtt{q}!\{m_j.T_i\}_{i\in I}, (\mathtt{q} \triangleleft \mathtt{p} : h) & \xrightarrow{\mathtt{p!q}:m_j} & \mathtt{p} : T_j, (\mathtt{q} \triangleleft \mathtt{p} : h.\langle \mathtt{p}, \mathtt{q}, m_j\rangle) & j \in I \\
\text{(InOut)} & \mathtt{q} : \mathtt{p}?\{m_i.T_i\}_{i\in I}, (\mathtt{p} \triangleright \mathtt{q} : h.\langle \mathtt{p}, \mathtt{q}, m_j\rangle) & \xrightarrow{\mathtt{p?q}:m_j} & \mathtt{p} : T_j, (\mathtt{p} \triangleright \mathtt{q} : h) & j \in I \\
\text{(OutOut)} & \mathtt{p} : \mathtt{q}!\{m_i.T_i\}_{i\in I}, (\mathtt{p} \triangleright \mathtt{q} : h) & \xrightarrow{\mathtt{p!q}:m_j} & \mathtt{p} : T_j, (\mathtt{p} \triangleright \mathtt{q} : \langle \mathtt{p}, \mathtt{q}, m_j\rangle.h) & j \in I \\
\text{(Transit)} & (\mathtt{p} \triangleright \mathtt{q} : h.\langle \mathtt{p}, \mathtt{q}, m\rangle), (\mathtt{q} \triangleleft \mathtt{p} : h) & \xrightarrow{\tau} & (\mathtt{p} \triangleright \mathtt{q} : h), (\mathtt{q} \triangleleft \mathtt{p} : \langle \mathtt{p}, \mathtt{q}, m\rangle.h) & \\
\text{(Par)} & \Delta_1 \xrightarrow{\ell} \Delta_1' & \Longrightarrow & \Delta_1, \Delta_2 \xrightarrow{\ell} \Delta_1', \Delta_2 &
\end{array}
$$

$(\mathtt{p} \triangleleft \mathtt{q} : h)$ (resp. $(\mathtt{p} \triangleright \mathtt{q} : h)$) stands for either $(\mathtt{p} \blacktriangleleft \mathtt{q} : h)$ (resp. $(\mathtt{p} \blacktriangleright \mathtt{q} : h)$) or $(\mathtt{p} \blacktriangleleft : h)$ (resp. $(\mathtt{p} \blacktriangleright : h)$)

Figure 3 Operational semantics of session cofigurations.

As seen above, local types do not represent a direct causality between sending m_1 and m_2 as the actions are done by different participants. There is however causality between the reception of m_1 and m_2 from the point-of-view of $\mathtt{p}$ – should the semantics be synchronous, this causality would be propagated to send operations.

3 Expressiveness of Multiparty Session Configurations

This section first defines the operational semantics of multiparty session types as *session configurations*. Then we define our notion of expressiveness, introducing the denotational semantics. Finally we show that (without asynchronous subtyping), expressive powers of all semantics are equivalent.

Semantics for sessions are transitions between *configurations* Δ: models of the state of a system through (i) a set of local types describing remaining actions to be performed by the participants and (ii) queues describing messages currently travelling in the networks. Semantics presented below are parametric w.r.t. the existence (and usage) of such queues.

3.1 Configuration semantics

The syntax of configurations (Δ) and queues (Q) is given below:

$$
\begin{array}{rcl@{\quad}rcl}
\Delta & ::= & \emptyset \mid \mathtt{p} : T, \Delta \mid Q, \Delta & h & ::= & \epsilon \mid \langle \mathtt{p}, \mathtt{q}, m\rangle.h \\
Q & ::= & (\mathtt{p} \blacktriangleleft \mathtt{q} : h) \mid (\mathtt{p} \blacktriangleright \mathtt{q} : h) \mid (\mathtt{p} \blacktriangleleft : h) \mid (\mathtt{p} \blacktriangleright : h) & & &
\end{array}
$$

Queues can be *output queues* $(\mathtt{p} \blacktriangleright \mathtt{q} : h), (\mathtt{p} \blacktriangleright : h)$ and store messages after they are produced by a participant – before they travel through the network – or *input queues* $(\mathtt{p} \blacktriangleleft \mathtt{q} : h), (\mathtt{p} \blacktriangleleft : h)$ and store messages before they are consumed by a participant – after they arrived from the network.

Queues can be linked to a single endpoint, the endpoint consuming messages for input queues, and the endpoint producing messages for output queues. They are written $(\mathtt{p} \blacktriangleleft : h)$ and $(\mathtt{p} \blacktriangleright : h)$ and are called *single queues*. Queues can also be labelled by two endpoints (source and destination of the message) and are in this case called *multiple queues* and written $(\mathtt{p} \blacktriangleleft \mathtt{q} : h)$ and $(\mathtt{p} \blacktriangleright \mathtt{q} : h)$.

The transition rules are given in Figure 3. In the following, the system will either (i) have no input (resp. output) queues, or (ii) one single input (resp. output) queue per participant

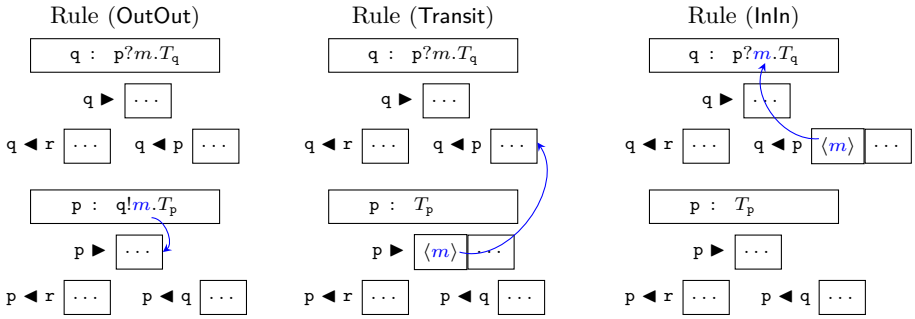


Figure 4 Illustration of several rules in the semantics $(M, 1)$.

Table 1 Rules used by the different semantics ϕ.

	$(0,0)$	$(0,1)$	$(0,M)$	$(1,0)$	$(1,1)$	$(1,M)$	$(M,0)$	$(M,1)$	(M,M)
(Com)	√								
(InIn)				√	√	√	√	√	√
(OutIn)				√			√		
(InOut)		√	√						
(OutOut)		√	√		√	√		√	√
(Transit)					√	√		√	√

and no multiple input (resp. output) queues, or (*iii*) one multiple input (resp. output) queues per pair of participants and no multiple input (resp. output) queues. A system with n participants with single input (resp. output) queues will have n input (resp. output) queues. A system with n participants with multiple input (resp. output) will have n^2 input (resp. output) queues. In the last rule, ℓ denotes a label which is either input ($\mathtt{p?q} : m_j$), output ($\mathtt{p!q} : m_j$), internal action (τ) or synchronisation ($\mathtt{pq} : m$).

A semantics ϕ is defined by a pair (I, O) representing the nature of the input and output queues of the system. I (resp. O) can be 0 (no input (resp. output) queues), 1 (single input (resp. output) queues), or M (multiple input (resp. output) queues). This effectively defines 9 different semantics using different sets of rules. They are summarised in Table 1.

Figure 4 illustrates the three rules used by semantics $(M, 1)$, i.e. a semantics with single output queues and multiple input queues. Rule (OutOut) consumes the action $\mathtt{q!}m$ of participant $\mathtt{p}$ to produce message $\langle \mathtt{p}, \mathtt{q}, m\rangle$ (noted as only $\langle m\rangle$ in the picture) in the output queue ($\mathtt{p} \blacktriangleright$). When the message reaches the end of the queue, it is dispatched to the input queue ($\mathtt{p} \blacktriangleleft \mathtt{q}$) through rule (Transit). Eventually, the message will be ready to be consumed by the action $\mathtt{p?}m$ of participant $\mathtt{q}$ by rule (InIn).

The $(0,0)$ semantics is the *synchronous semantics*. The three semantics $(1, \_)$ are called the *single-input semantics* or *unsafe semantics* and the six other ones are called *safe semantics*. These names come from Proposition 11. We say that $\Delta_1 \xrightarrow{\ell} \Delta_2$ through semantics ϕ when $\Delta_1 \xrightarrow{\ell} \Delta_2$ is derived with rules belonging to ϕ (according to Table 1).

In the following, we consider that two systems are different if the possible sequences of actions for *one* participant differ. We only consider the order of actions happening locally. We will compare configuration traces, which are collections of local traces.

An event e is either a send event $\mathtt{p!}m$ or a receive event $\mathtt{p?}m$. For a participant, sending corresponds either to a communication (synchronous semantics), or putting a message in its own output queue ($(\_, 1)$ and $(\_, M)$), or putting a message in the target input queue $(\_, 0)$.

▶ **Definition 2** (Configuration traces). A configuration trace σ is a mapping from participants to finite sequences of events: $\sigma(\mathtt{r}) = (e)_{n \leq N}$ for $N \in \mathbb{N}$. We use ϵ for the empty sequence. A participant $\mathtt{r}$ is in the *domain* of σ if $\sigma(\mathtt{r}) \neq \epsilon$. The length of a trace σ is the sum of the length of the sequences $\sigma(\mathtt{r})$ for all $\mathtt{r}$ in its domain. We say $\sigma \leq \sigma'$ when $\forall \mathtt{r}$, $\sigma(\mathtt{r})$ is a sequence prefix of $\sigma'(\mathtt{r})$.

The relation between traces and configuration is given by the relation $\Delta \rightsquigarrow_{\phi}^{\sigma} \Delta'$ meaning Δ *executes trace* σ_0 *to* Δ *for semantics* ϕ defined with:

1. For any configuration Δ and semantics ϕ, $\Delta \rightsquigarrow_{\phi}^{\sigma_0} \Delta$ where σ_0 is defined by: for all roles $\mathtt{r}$, $\sigma_0(\mathtt{r}) = \epsilon$.
2. For any configurations Δ, Δ_1, Δ_2, any trace σ, any label ℓ, and any semantics ϕ, if $\Delta \rightsquigarrow_{\phi}^{\sigma} \Delta_1$ and if $\Delta_1 \xrightarrow{\ell} \Delta_2$ through ϕ, then we define $\Delta \rightsquigarrow_{\phi}^{\sigma'} \Delta_2$ as follows:
   - **a.** if $\ell = \mathtt{p}!\mathtt{q} : m_j$, then σ' is defined by: $\sigma'(\mathtt{p}) = \sigma(\mathtt{p}).\mathtt{q}!m_j$ and $\sigma'(\mathtt{r}) = \sigma(\mathtt{r})$ for $\mathtt{r} \neq \mathtt{p}$.
   - **b.** if $\ell = \mathtt{p}?\mathtt{q} : m_j$, then σ' is defined by: $\sigma'(\mathtt{q}) = \sigma(\mathtt{q}).\mathtt{p}?m_j$ and $\sigma'(\mathtt{r}) = \sigma(\mathtt{r})$ for $\mathtt{r} \neq \mathtt{p}$.
   - **c.** if $\ell = \mathtt{pq} : m_j$, then σ' is defined by: $\sigma'(\mathtt{p}) = \sigma(\mathtt{p}).\mathtt{q}!m_j$, $\sigma'(\mathtt{q}) = \sigma(\mathtt{q}).\mathtt{p}?m_j$ and $\sigma'(\mathtt{r}) = \sigma(\mathtt{r})$ for $\mathtt{r} \notin \{\mathtt{p}, \mathtt{q}\}$.
   - **d.** if $\ell = \tau$, then $\sigma' = \sigma$.

A trace σ is in the *trace set* of a configuration Δ for a semantic ϕ, written $\sigma \in \mathcal{T}_{\phi}(\Delta)$, (we sometimes write Δ *has trace* σ *for semantics* ϕ) whenever there exist Δ' s.t. $\Delta \rightsquigarrow_{\phi}^{\sigma} \Delta'$. The trace set of a global type G for the semantics ϕ is the trace set for semantics ϕ of configuration $\delta(G)$ defined by $\delta(G) = \mathtt{r}_1 : T_1 \ldots, \mathtt{r}_n : T_n, Q_1, \ldots, Q_n$ where $\mathtt{r}_1, \ldots, \mathtt{r}_n$ are the roles involved in G, $T_i = G\upharpoonright(\mathtt{r}_i)$, and Q_i are all empty ϵ and correspond to ϕ. A *terminated trace* of a global type G for the semantics ϕ is a trace σ s.t. $\delta(G) \rightsquigarrow_{\phi}^{\sigma} \Delta$ where $\Delta \not\rightarrow$. A *completed trace* of a global type G for the semantics ϕ is a trace σ s.t. $\delta(G) \rightsquigarrow_{\phi}^{\sigma} 0$ where $0 = \mathtt{r}_1 : \mathtt{end}, \ldots, \mathtt{r}_n : \mathtt{end}, Q_1, \ldots, Q_n$ where Q_i are all ϵ. A completed trace is terminated.

▶ **Example 3** (Configuration). Let $\Delta_e = \{\mathtt{p} : \mathtt{q}!m_1.\mathtt{r}!m_3, \mathtt{q} : \mathtt{p}?m_1.\mathtt{r}!m_2, \mathtt{r} : \mathtt{q}?m_2.\mathtt{p}?m_3\}$ (cf. Example 1). The initial configuration from G_e for $(0, 0)$ is Δ_e, the one for $(M, 1)$ is $\Delta_e, (\mathtt{p} \blacktriangleright: \epsilon), (\mathtt{q} \blacktriangleright: \epsilon), (\mathtt{r} \blacktriangleright: \epsilon), (\mathtt{p} \blacktriangleleft \mathtt{q} : \epsilon), (\mathtt{p} \blacktriangleleft \mathtt{r} : \epsilon), (\mathtt{q} \blacktriangleleft \mathtt{p} : \epsilon), (\mathtt{q} \blacktriangleleft \mathtt{r} : \epsilon), (\mathtt{r} \blacktriangleleft \mathtt{p} : \epsilon), (\mathtt{r} \blacktriangleleft \mathtt{q} : \epsilon)$. Both configurations can evolve along the terminated trace $\sigma_e : \mathtt{p} \mapsto \mathtt{q}!m_1.\mathtt{r}!m_3, \mathtt{q} \mapsto \mathtt{p}?m_1.\mathtt{r}!m_2, \mathtt{r} \mapsto \mathtt{q}?m_2.\mathtt{p}!m_3$ even if non-terminated traces are different; for instance $\sigma_t : \mathtt{p} \mapsto \mathtt{q}!m_1, \mathtt{q} \mapsto \emptyset, \mathtt{r} \mapsto \emptyset$ is a valid trace from G_e by $(M, 1)$ and not by $(0, 0)$.

3.2 Expressiveness via denotational semantics

We can extract from Definition 2 *a denotation* of a global type G, w.r.t a particular semantics, as the set of terminated traces of G. We can compare, for a given type G, the terminated traces of G for two different semantics. As sessions ensure the local interaction follows an expected behaviour, local traces are strongly constrained by the semantics. This observation is still useful for two reasons: (i) it establishes a distinction between safe semantics which prevents deadlocks from arising and unsafe semantics, and (ii) further operations (§ 4, 5.1 and 5.2) on types will remove this constraint. Secondly, we can associate, to a type G containing n participants, the languages $\{L_i\}_{i \leq n}$ corresponding to the local traces in the set of terminated traces for G. We can then consider the *expressive power* of ϕ as the set of all languages obtainable for all possible G with ϕ. We define ϕ_1 has greater expressive power than ϕ_2 if all languages in the expressive power of ϕ_2 are in the expressive power of ϕ_1.

▶ **Definition 4** (Denotation of a type under a semantics). We define the *denotation* of global type G under semantics ϕ, noted $\mathbf{D}(G, \phi)$, as the set of all terminated traces from G w.r.t. ϕ.

▶ **Definition 5** (Progress). We say that ϕ ensures progress if for all G, $\mathbf{D}(G,\phi)$ contains only completed traces.

If role $\mathtt{r}$ appears in G, $\mathbf{D}(G,\phi,\mathtt{r})$ is the set of all local traces for $\mathtt{r}$ obtained from terminated traces of $\mathbf{D}(G,\phi)$, that is $\mathbf{D}(G,\phi,\mathtt{r}) = \{\sigma(\mathtt{r}) | \sigma \in \mathbf{D}(G,\phi)\}$.

▶ **Definition 6** (Expressive power of a semantics). We define the *expressive power* of semantics ϕ as follows: $\{\mathbf{D}(G,\phi,\mathtt{r}) \mid \mathtt{r} \in G \text{ and } G \text{ well-formed}\}$, that is, the collection of all languages of local traces corresponding to terminated traces from well-formed global types.

3.3 Expressiveness results (without subtyping)

The first theorem (Theorem 10) states that all safe semantics give the same denotations: adding non-single-input queues has no influence on denotations. We first prove confluence of semantics by stating that we can always complete a trace of any semantics by using synchronous semantics.

▶ **Lemma 7** (Confluence of trace semantics). *Let ϕ_1 be a safe semantics and G a well-formed global type. If $\delta(G) \rightsquigarrow^{\sigma}_{\phi_1} \Delta$, there exists Δ', σ' s.t. $\delta(G) \rightsquigarrow^{\sigma.\sigma'}_{\phi_1} \Delta'$ and $\delta(G) \rightsquigarrow^{\sigma.\sigma'}_{(0,0)} \Delta'$*

▶ **Definition 8** (Prefix). Let T be a type, σ a trace and $\mathtt{r}$ a participant. The prefix relation $\sigma(\mathtt{r}) <_{\mathtt{p}} T$ is defined as: (1) $\epsilon <_{\mathtt{p}} T$; (2) $\mathtt{p}!m_j.\sigma <_{\mathtt{p}} \mathtt{q}!\{T_i\}_{i\in I}$ and $\sigma <_{\mathtt{p}} T_j$; and (3) $\mathtt{p}?m_j.\sigma <_{\mathtt{p}} \mathtt{q}?\{T_i\}_{i\in I}$ and $\sigma <_{\mathtt{p}} T_j$.

▶ **Lemma 9** (Session fidelity). *Let ϕ be a safe semantics and G a well-formed global type. If $\delta(G) \rightsquigarrow^{\sigma}_{\phi} \Delta$, $\mathtt{r} \in G$, then $\sigma(\phi) <_{\mathtt{p}} G\upharpoonright(\mathtt{r})$.*

▶ **Theorem 10** (Expressiveness of safe semantics). *For any G, the sets $\mathbf{D}(G,\phi)$ for all safe ϕ are the same.*

Proof. Done by proving that all safe semantics are equivalent to the synchronous one: local traces stay the same, as they are constrained by the initial global types. Suppose σ_1 is a completed trace for ϕ_1. By using Lemma 7 there exists σ_1' s.t. $\sigma_1.\sigma_1'$ is a trace for ϕ_1 and $(0,0)$. By Lemma 9, σ_1' is ϵ. It follows that σ_1 is a completed trace for $(0,0)$, thus a completed trace for ϕ_2. ◀

However, unsafe semantics are not comparable with the others, as they lead to deadlocks.

▶ **Proposition 11** (Single input deadlock). Unsafe semantics do not ensure progress.

Proof. Consider a global type $G_e = \mathtt{p} \to \mathtt{q} : m_1.\mathtt{q} \to \mathtt{r} : m_2.\mathtt{p} \to \mathtt{r} : m_3$ with the $(1,0)$ semantics (same reasoning applies to $(1,0)$ and $(1,M)$). After the sequence $\mathtt{p}!\mathtt{q} : m_1.\mathtt{p}!\mathtt{r} : m_3.\mathtt{p}?\mathtt{q} : m_1.\mathtt{q}!\mathtt{r} : m_2$, $\mathtt{r}$ type is $\mathtt{q}?m_2.\mathtt{p}?m_3.\mathtt{end}$ and its queue is $(\mathtt{r} \blacktriangleleft \bullet : \langle \mathtt{p}, \mathtt{r}, m_3\rangle\langle \mathtt{q}, \mathtt{r}, m_2\rangle)$ meaning the system can no longer proceed: $\mathtt{r}$ expects m_2 then m_3 but the queue offers m_3 then m_2. ◀

▶ **Proposition 12** (Regularity). *The languages in expressive power of safe semantics are regular.*

Proof. Suppose G is a session type containing participant $\mathtt{p}$. We prove that the traces accepted by local $T = G\upharpoonright(\mathtt{r})$ by induction on T. We present only interesting cases.

- If $T = \mathtt{q}?\{m_i.T_i\}_{i\in I}$, then the possible completed traces $\mathtt{q}?l_j.t$ all start with one $\mathtt{q}?l_j$ and follows by an accepted trace t_j of T_j. By induction, the language L_j of all t_j is regular. Thus the accepted language is the sum, for all j of the traces $\mathtt{q}?l_j.L_j$, and is regular.

- If $T = \mu \mathtt{t}.T'(\mathtt{t})$ we check the number of occurrences of `end` in $T'(\mathtt{t})$. If `end` does not appear, the accepted language of T is $\emptyset$. Otherwise, by masking about the recursion token `t`, we can see the accepted traces of T' as a sum of language ΣL_k, each language L_k corresponding of one branch of T'. There is at most one branch ending with `t`, suppose it is the branch corresponding to L_1, it means the accepted traces of T are $(L_1) * .(\Sigma_{k \neq 1} L_k)$, which is regular. ◀

▶ Remark (Asymmetry of expressiveness). Progress requires input queues to be multiple (or no input queues at all), but is independent from output queues. Output queues do not affect expressiveness as they cannot block endpoints: if there is an input queue, the former can always unload its content in the latter, if there is none, as the order of messages in the output queue matches the order of the session, session type soundness ensures progress. Without input queues, *session fidelity* [11] is required, for instance consider the configuration $\mathtt{r} : \mathtt{p}!m_1.\mathtt{q}!m_2.\mathtt{end}, \mathtt{p} : \mathtt{q}!m_3.\mathtt{r}?m_1.\mathtt{end}, \mathtt{q} : \mathtt{r}?m_2.\mathtt{p}?m_1.\mathtt{end}$. Terminated traces are different for $(0, 0)$ and $(0, 1)$ (in this case, the system is blocked as it would require m_2 to be sent before m_1) and for $(0, M)$ (the system can proceed to a completed state); however, the initial configuration does not correspond to a global type.

4 Expressiveness of subtyping

As described in the previous section, the mechanisms of session maintain an order over local components traces: actions performed by the participants happen in the exact order expected by types. As a consequence, if transport structures change the order of arrival of messages from different sources – a realistic assumption, this condition can lead to deadlocks.

Implementations of session types [22, 18] sometimes enforce *flexibility* for the endpoint application: for instance, by allowing two outputs expected to be sent sequentially to be performed in any order. This flexibility is represented formally by the use of *subtyping*, as in [16, 6, 17], describing transformations on types by switching pairs of consecutive actions. We study input-input and output-output flexibility which make it possible to exchange two consecutive actions of the same nature. Input-output (resp. output-input) flexibility allows one output (resp. input) to be performed before previously expected inputs *and* outputs.

4.1 Subtyping rules

As explained above, input-input flexibility offers the possibility for the second input in a sequence of two consecutive inputs to be executed first. For instance, input flexibility allows type $\mathtt{p}?m_1.\mathtt{q}?m_2.\mathtt{p}!m_3.\mathtt{end}$ to be converted into $\mathtt{q}?m_2.\mathtt{p}?m_1.\mathtt{p}!m_3.\mathtt{end}$. Consider the following types:

$$T = \mathtt{p}? \begin{cases} m_{11}.\mathtt{q}?m_2.\mathtt{p}!m_3.\mathtt{end} \\ m_{12}.\mathtt{q}?m_2.\mathtt{p}!m_4.\mathtt{end} \end{cases} \qquad T' = \mathtt{q}?m_2.\mathtt{p}? \begin{cases} m_{11}.\mathtt{p}!m_3.\mathtt{end} \\ m_{12}.\mathtt{p}!m_4.\mathtt{end} \end{cases}$$

In T, the first input is branching and models a program reacting to two different messages from `p`, either m_{11} or m_{12}. In both branches, a message m_2 from `q` is expected. One would expect an input-input flexible program to be able to accept message m_2 if it arrives first, which would mean converting T to T', which means T' is a subtyping of T.

For a subtyping definition, we introduce the input and output contexts. They are parametered with the name of the participant of the action being permuted, as we do not exchange two actions with the same participant. An input (resp. output) context is a term formed only of branched input (resp. output) actions, seen as tree where each branch finishes

with a hole. Input-output and output-input contexts contain both type of actions, they only differ in the name verification. Flexibility is defined by the subtyping rules, which realise the possible exchanges in branched types.

▶ **Definition 13** (Input/Output contexts and Subtyping)**.** Input and input/output type contexts are defined by:

$$\mathbb{C}^{\mathtt{q}}_{\mathsf{I}} ::= [\,] \mid \mathtt{p}?\{m_i.\mathbb{C}^{\mathtt{q}}_{\mathsf{I}}\}_{i\in I}\ (\mathtt{p}\neq\mathtt{q}) \qquad \mathbb{C}^{\mathtt{q}}_{\mathsf{IO}} ::= [\,] \mid \mathtt{p}?\{m_i.\mathbb{C}^{\mathtt{q}}_{\mathsf{IO}}\}_{i\in I} \mid \mathtt{r}!\{m_i.\mathbb{C}^{\mathtt{q}}_{\mathsf{IO}}\}_{i\in I}\ (\mathtt{r}\neq\mathtt{q})$$
$$\mathbb{C}^{\mathtt{q}}_{\mathsf{O}} ::= [\,] \mid \mathtt{q}!\{m_i.\mathbb{C}^{\mathtt{q}}_{\mathsf{O}}\}_{i\in I}\ (\mathtt{p}\neq\mathtt{q}) \qquad \mathbb{C}^{\mathtt{q}}_{\mathsf{OI}} ::= [\,] \mid \mathtt{p}!\{m_i.\mathbb{C}^{\mathtt{q}}_{\mathsf{OI}}\}_{i\in I}\ (\mathtt{p}\neq\mathtt{q}) \mid \mathtt{r}?\{m_i.\mathbb{C}^{\mathtt{q}}_{\mathsf{OI}}\}_{i\in I}$$

Subtyping $\leq$ is coinductively defined by the following rules:

$$(\mathsf{II})\ \frac{\forall(i,k), T_i \leq \mathtt{q}?m_k.\mathbb{C}^{\mathtt{p}}_{\mathsf{I}}[T'_i] \quad \mathtt{q}\neq\mathtt{p}}{\mathtt{p}?\{m_i.T_i\}_{i\in I} \leq \mathtt{q}?\{m_k.\mathbb{C}^{\mathtt{q}}_{\mathsf{I}}[\mathtt{p}?\{T'_i\}_{i\in I}]\}_{k\in K}} \qquad (\mathsf{OO})\ \frac{\forall(i,k), T_i \leq \mathtt{q}!m_k.\mathbb{C}^{\mathtt{p}}_{\mathsf{O}}[T'_i] \quad \mathtt{q}\neq\mathtt{p}}{\mathtt{p}!\{m_i.T_i\}_{i\in I} \leq \mathtt{q}!\{m_k.\mathbb{C}^{\mathtt{q}}_{\mathsf{O}}[\mathtt{p}!\{T'_i\}_{i\in I}]\}_{k\in K}}$$

$$(\mathsf{IO})\ \frac{\forall(i,k), T_i \leq \mathtt{q}!m_k.\mathbb{C}^{\mathtt{p}}_{\mathsf{IO}}[T'_i] \quad \mathtt{q}\neq\mathtt{p}}{\mathtt{p}!\{m_i.T_i\}_{i\in I} \leq \mathtt{q}?\{m_k.\mathbb{C}^{\mathtt{q}}_{\mathsf{IO}}[\mathtt{p}!\{T'_i\}_{i\in I}]\}_{k\in K}} \qquad (\mathsf{OI})\ \frac{\forall(i,k), T_i \leq \mathtt{q}!m_k.\mathbb{C}^{\mathtt{p}}_{\mathsf{OI}}[T'_i] \quad \mathtt{q}\neq\mathtt{p}}{\mathtt{p}?\{m_i.T_i\}_{i\in I} \leq \mathtt{q}!\{m_k.\mathbb{C}^{\mathtt{q}}_{\mathsf{IO}}[\mathtt{p}?\{T'_i\}_{i\in I}]\}_{k\in K}}$$

Subtyping is extended on configurations. To describe semantics with subtyping, we define a *subtyping policy* $\mathcal{P}$ as a subset of $\{\mathsf{OO},\mathsf{II},\mathsf{OI},\mathsf{IO}\}$ abiding to $\mathsf{OO}\in\mathcal{P}\Rightarrow\mathsf{OI}\in\mathcal{P}$ and $\mathsf{OI}\in\mathcal{P}\Rightarrow\mathsf{OO}\in\mathcal{P}$. We use the notation $T_1\leq_{\mathcal{P}} T_2$ to state that $T_1\leq T_2$ is derivable using only rules in $\mathcal{P}$. We consider semantics ϕ associated to a subtyping policy $\mathcal{P}$, which are obtained by adding the following rule:

$$(\mathsf{Sub}:\mathcal{P}) \qquad \Delta_1 \xrightarrow{l} \Delta_2 \quad \Delta'_1 \leq_{\mathcal{P}} \Delta_1 \quad \Longrightarrow \quad \Delta'_1 \xrightarrow{l} \Delta_2$$

Subtyping makes it possible to first perform an action that is present in the branches of all possible behaviours. (II) performs inputs from different senders in any order; (OO) performs an output before other outputs to different receivers, (IO) allows an output to be performed before other actions with different participants, and (OI) allows an input to be performed before other actions with different participants.

4.2 Progress and expressiveness of flexibility

Flexibility introduces deadlock under the synchronous semantics. For instance, consider the global type $\mathtt{r}\to\mathtt{q}:m.\mathtt{q}\to\mathtt{p}:m_1.\mathtt{r}\to\mathtt{p}:m_2.\mathtt{end}$. In one application of (IO) on the initial configuration, we reach configuration $(\mathtt{r}:\mathtt{q}!m.\mathtt{p}!m_2,\mathtt{q}:\mathtt{p}!m_1.\mathtt{r}?m_2,\mathtt{p}:\mathtt{q}?m_1.\mathtt{r}!m_2)$ which is locked for synchronous semantics, and we have no means to retrieve initial configuration. Proposition 15 and Table 2 describe which association of a semantics ϕ and a set of subtyping rules $\mathcal{P}$ avoid deadlocks. A subtyping policy $\mathcal{P}$ is *safe* w.r.t. a semantics ϕ if the system $\mathcal{P}$ ensure progress under ϕ.

▶ **Lemma 14.** *$\emptyset$, $\{\mathsf{OO}\}$ and $\{\mathsf{II}\}$ and $\{\mathsf{OI},\mathsf{IO}\}$ are safe w.r.t. all safe semantics.*
- *$\{\mathsf{IO}\}$ is safe w.r.t. all non-synchronous semantics.*
- *$\{\mathsf{OI}\}$ is unsafe w.r.t. all semantics.*

▶ **Proposition 15** (Safe subtyping)**.** Safety for subtyping policy and semantics is given by Table 2: "$\surd$" represents a safe semantics, and "$\times$" an unsafe one.

For a given semantics, one can use subtyping to complete traces that are not possible without it. As a simple example consider $\mathtt{p}\to\mathtt{q}:m_1.\mathtt{p}\to\mathtt{r}:m_2$, trace $\mathtt{r}!m_2.\mathtt{q}!m_1$ is accepted with OO-subtyping but cannot be accepted otherwise. We use $\mathbf{D}(G,\mathcal{P},\phi)$ to represent the denotation of type G under semantics ϕ and subtyping rules $\mathcal{P}$. Below Proposition 16 confirms that subtyping actually changes the denotations of types. Theorem 17 states that subtyping accept traces beyond the regular languages.

Table 2 Safe subtyping with respect to semantics.

	$(0,0)$	$(0,1)$	$(0,D)$	$(1,0)$	$(1,1)$	$(1,D)$	$(D,0)$	$(D,1)$	(D,D)
$\emptyset$	√	√	√	×	×	×	√	√	√
II	√	√	√	√	√	√	√	√	√
OO	√	√	√	×	×	×	√	√	√
IO	×	√	√	√	√	√	√	√	√
OI	×	×	×	×	×	×	×	×	×
IO, OI	√	√	√	√	√	√	√	√	√

▶ **Proposition 16** (Denotations in presence of subtyping). *If ϕ is safe and $\mathcal{P}_1 \subsetneq \mathcal{P}_2$,*
1. *for all projectable global type G, $\mathbf{D}(G, \mathcal{P}_1, \phi) \subseteq \mathbf{D}(G, \mathcal{P}_1, \phi)$*
2. *there exists a projectable global type G, $\mathbf{D}(G, \mathcal{P}_1, \phi) \subsetneq \mathbf{D}(G, \mathcal{P}_1, \phi)$*

▶ **Theorem 17** (Expressive power of subtyping). *The expressive power of subtyped sessions is strictly greater than the expressive power of standard sessions.*

Proof. We prove that the expressive power of subptyped sessions contained non-regular languages. Consider the type $G = \mu\mathtt{t}.\mathtt{p} \to \mathtt{q}.\mathtt{p} \to \mathtt{r}.\mathtt{t}$. Its projection on $\mathtt{p}$ is $T = \mu\mathtt{t}.(\mathtt{q}!.\mathtt{r}!.\mathtt{t} + \mathtt{q}.\mathtt{end})$. Although it is of no importance for the following, we can establish the possible completed traces from T by the safe semantics $(\emptyset, \emptyset)$ is $(\mathtt{q}!.\mathtt{r}!)^*.\mathtt{q}!$. (1) With the semantics $(\emptyset, \{\mathsf{OO}\})$, it is easy to see the language L of possible completed traces are all the words obtain by shuffling $\mathtt{q}!^{n+1}$ and $\mathtt{r}!^n$: all completed traces that contains n $!\mathtt{q}$ contains exactly $(n-1)$ $\mathtt{r}!$ and type permutations allow us to move any $\mathtt{r}!$ leftwards in any trace and stay inside the completed trace set – thanks to the OO rule. (2) The shuffling of $\mathtt{q}!^{n+1}$ and $\mathtt{r}!^n$ is not regular. It follows from Proposition 12 that there does not exist a type G s.t. the language of possible traces for $\mathtt{p}$ in $G\upharpoonright(\mathtt{p})$ is L. ◀

5 Expressiveness of parallel and interruptible sessions

5.1 Influence of parallel composition

In the previous section, the reduction rules are updated with subtyping, giving flexibility to the local endpoint. Similar behaviour can also be reached by adding in the syntax a parallel operator explicitly stating that two actions can be performed in any order. We consider in the following, *parallel sessions* which are sessions with the additional constructs for parallel composition $G_1 \mid G_2$ and $T_1 \mid T_2$. Related projection and semantics rules are standard and can be found in [11].

As expected, adding parallel composition of actions, leads to irregularity of the safe semantics. Both subtyping rule and parallel syntax can be used to get rid of potential deadlocks. Parallel composition allows one session designer to precisely describe which actions are unordered whereas subtyping is a global policy for permutation. As a result, parallel sessions are more expressive than subtyping sessions, giving a finer control over which actions can be exchanged.

▶ **Proposition 18** (Parallel).
1. Expressive power of parallel sessions contains irregular languages.
2. Parallel sessions have a strictly greater expressive power than subtyping sessions.

Proof. 1. The language of completed traces accepted at $\mathtt{p}$ for the type $\mu\mathtt{t}.(\mathtt{p} \to \mathtt{q}.\mathtt{t} + \mathtt{p} \to \mathtt{q}.\mathtt{end} \mid \mathtt{p} \to \mathtt{r}.\mathtt{end})$ is not regular – and is a particular shuffling of $\mathtt{q}!^n$ and $\mathtt{r}!^n$.

2. For every G without parallel and every safe combination of ϕ and $\mathcal{P}$, we can find G' which has exactly the same trace set. G' is obtained from G by putting in parallel consecutive events from G w.r.t. $\mathcal{P}$. Moreover, some types containing both parallel and sequences such as $(\mathtt{p} \to \mathtt{q}_1.\mathtt{p} \to \mathtt{q}_2 \mid \mathtt{p} \to \mathtt{r}_1.\mathtt{p} \to \mathtt{r}_2)$ cannot be expressed with subtyping only. ◀

5.2 Expressiveness of Interruptible Session Types

Interruptible sessions introduce a mechanism for participants to exceptionally exit blocks of interactions; we study here the resulting gain in expressiveness. Interruptible session types are presented in [8, 12]: in the global type syntax, particular ranges of actions (called *scopes*) can be interrupted at any time by a participant; a special message is broadcasted to all participants of the scope. As soon as one of them is notified of the interruption, it gives up any action related to this scope. Interruptions have been included in protocol language *Scribble* [22] because it was needed to represent usecases in [1], see [12]. We prove here that this inclusion is necessary, and that such protocols cannot be described with standard sessions.

As a simple example of the interrupt, consider:

$$G = \{\!|\mathtt{r} \to \mathtt{p} : m.(\mu \mathtt{t}.\mathtt{p} \to \mathtt{q} : m_1.\mathtt{q} \to \mathtt{p} : m_2.\mathtt{t})|\!\}^{\mathtt{c}}\langle i \text{ by } \mathtt{r}\rangle; \mathtt{q} \to \mathtt{r} : a.\mathtt{end}$$

G is a type consisting of one interruptible scope $\mathtt{c}$. It starts with message m from $\mathtt{r}$ to $\mathtt{p}$, which initiates a loop of messages m_1 and m_2 between $\mathtt{p}$ and $\mathtt{q}$. At any time during this loop, $\mathtt{r}$ can decide to stop it by raising an interruption: message i is sent to both $\mathtt{p}$ and $\mathtt{q}$ which are expected to stop interacting with each other as soon as they receive it. After interruption, the session then resumes by a message a from $\mathtt{q}$ to $\mathtt{r}$.

Interruptible session types are standard session types with the addition of scope constructions. $\{\!|G|\!\}^{\mathtt{c}}\langle l \text{ by } \mathtt{r}\rangle; G'$ is the global type composed of one scope name $\mathtt{c}$ encompassing the type G. At any time, progress inside G can be interrupted by participant $\mathtt{r}$ with a special interrupt message carrying label l and $\mathtt{Eend}$ stands for a exceptionally ended scope. After G is finished - either normally or exceptionally, the protocol continues as G'. Each scope $\mathtt{c}$ is associated to a set of participants involved through the mapping Γ. Such information allows the semantics to notify each participant when an exceptional behaviour arises. Additional projection rules needed for interruptible scopes, projection remembers whether it projects on the name which can interrupt the scope or not, resulting in two different constructs for interruptible local types:

$$\begin{array}{rcl} \{\!|G|\!\}^{\mathtt{c}}\langle l \text{ by } \mathtt{r}\rangle; G'\upharpoonright(\mathtt{r}) & = & \{\!|G\upharpoonright(\mathtt{r})|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{r}?l\rangle; G'\upharpoonright(\mathtt{r}) \\ \{\!|G|\!\}^{\mathtt{c}}\langle l \text{ by } \mathtt{r}'\rangle; G'\upharpoonright(\mathtt{r}) & = & \{\!|G\upharpoonright(\mathtt{r})|\!\}^{\mathtt{c}} \triangleleft \langle \mathtt{r}'!l\rangle; G'\upharpoonright(\mathtt{r}) \quad \text{when } \mathtt{r} \in G \qquad \text{otherw.} \quad G'\upharpoonright(\mathtt{r}) \end{array}$$

Excerpt of configuration semantics for interruptible sessions (details are in [8, 12]) is given below through the use of evaluation contexts $\mathbb{C}^{\mathtt{c}}$ which has a hole in $\{\!|\_|\!\}$ and after the sequential composition (formally defined in [8, 12]).

$$\begin{array}{ll} (\mathsf{EOut}) & \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|T|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{r}?l\rangle; T'], \mathtt{r}_1 : h, \ldots, \mathtt{r}_n : h \\ & \quad \to \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|\mathtt{Eend}|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{r}?l\rangle; T'], \mathtt{r}_1 : \langle \mathtt{c}^{\mathtt{I}}, \mathtt{r}, \mathtt{r}_1, l\rangle.h, \ldots, \mathtt{r}_n : \langle \mathtt{c}^{\mathtt{I}}, \mathtt{r}, \mathtt{r}_n, l\rangle.h \\ (\mathsf{EIn}) & \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|T|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{q}?l\rangle; T'], \mathtt{r} : h.\langle \mathtt{c}^{\mathtt{I}}, \mathtt{q}, \mathtt{r}, l\rangle.h \to \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|\mathtt{Eend}|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{q}?l\rangle; T'], \mathtt{r} : h \\ (\mathsf{Disc}) & \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|\mathtt{Eend}|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{q}?l\rangle; T'], \mathtt{r} : \langle \mathtt{c}_1, \mathtt{q}, \mathtt{r}, l\rangle.h \to \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|\mathtt{Eend}|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{q}?l\rangle; T'], \mathtt{r} : h \\ (\mathsf{EDisc}) & \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|\mathtt{Eend}|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{q}?l\rangle; T'], \mathtt{r} : \langle \mathtt{c}_1^{\mathtt{I}}, \mathtt{q}, \mathtt{r}, l\rangle.h \to \mathtt{r} : \mathbb{C}^{\mathtt{c}_0}[\{\!|\mathtt{Eend}|\!\}^{\mathtt{c}} \triangleright \langle \mathtt{q}?l\rangle; T'], \mathtt{r} : h \end{array}$$

In this framework, in-transit messages contains scope information $\mathtt{c}$, it allows interrupt messages to exit the right scope. In (EOut), participant $\mathtt{r}$ decides to raise an interruption of scope $\mathtt{c}$ and continues as T' but remembers that scope $\mathtt{c}$ was exited exceptionally; interruption

messages are broadcasted to all participants present in scope $\mathtt{c}$. In (EIn) a participant $\mathtt{r}$ executing actions in scope $\mathtt{c}$ receives an interrupt messages from $\mathcal{Q}$, and immediately exits $\mathtt{c}$. Rule (Disc) (resp. rule (EDisc)) is used to discard incoming standard (resp. interruption) messages to already-exited scopes.

Theorem 20 claims that session languages with interruptions have greater expressiveness. Denotations resulting from interruptible session types cannot be obtained by use of parallel, choice and flexibility subtyping.

▶ **Lemma 19.** *Expressive powers of standard, parallel and subtyped sessions do not contain languages of the form $a^n.b^k$ with $k \leq n$.*

▶ **Theorem 20** (Expressiveness of interruptible sessions)**.**

1. *Interruptible sessions have a strictly greater expressive power than sessions.*
2. *Interruptible sessions have a different expressive power than parallel sessions.*
3. *Interruptible sessions have a different expressive power than subptyped sessions.*

Proof. Standard session behaviours are included into interruptible sessions. Separation proofs are based on, stating that only interruptible sessions allows trace languages of the form $a^n.b^k$ with $k \leq n$. Lemma 19 is proved by stating that without interruptions, a denotation containing traces where actions a all appear before actions b is such that the number of a and b are independent (coming from the unfoldings of two different recursions). Interruptible type $\mu\mathtt{t}.\{\!|\mathtt{p} \to \mathtt{q} : m_1.\mathtt{t}|\!\}^{\mathtt{c}}\langle i \text{ by } \mathtt{q}\rangle; \mathtt{q} \to \mathtt{p} : m_2.\mathtt{end}$, is a loop of nested scopes. Messages m_1 are continuously received by $\mathtt{q}$ (unfolding the recursion once at each reception) until an interruption is raised, using rule (EOut). Afterwards, $\mathtt{q}$ discards further incoming m_1 messages with rule (Disc) and the only possible actions is sending of m_2 messages. The number of m_2 messages to be sent corresponds to the number of unfoldings done while receiving m_1, and thus is lower than the number of messages m_1 received. As a consequence, the expressive power of interruptible sessions contains the languages $a^n.b^k$ with $k \leq n$, which does not appear in the expressive power of standard, parallel and subtyped sessions. Interruptible sessions are not strictly more expressive than parallel and subtyped sessions. The languages corresponding to types $\mu\mathtt{t}.(\mathtt{p} \to \mathtt{q}.\mathtt{t} + \mathtt{p} \to \mathtt{q}.\mathtt{end} \mid \mathtt{p} \to \mathtt{r}.\mathtt{end})$ from Proposition 18 and $\mu\mathtt{t}.\mathtt{p} \to \mathtt{q}.\mathtt{p} \to \mathtt{r}.\mathtt{t}$ from Theorem 17 are not in the expressive power of interruptible sessions, which does not contain shufflings. ◀

6 Related Works

There is a vast literature on expressiveness studies for process calculi; we refer to [20] for a survey (see also [21, § 2.3]). Our work is original (i) we study expressiveness of *types*, based on the language theory; (ii) we compare the design choices of the network (queue) topology (§ 3); the local permutations (§ 4); and the type constructs (parallel in § 5.1 and and interrupt in § 5.2); and (iii) our notion of expressiveness is based on denotational and operational semantics: we compare completed traces of local actions induced by a global type. As far as we have known, this is the first work to define and investigate expressiveness based on denotations and languages made by traces of concurrecy types.

Our concurrent model stems from a previous work [2] in which networks are modeled as configurations, i.e. collections of types and queues. The model used in [2] is *multisession* and uses routing information updated at runtime to maintain network topology. This feature has been removed for the sake of clarity, as it has little impact on expressiveness.

The first part of our work, focusing on expressiveness on different queue configurations, is inspired by [14] where they studied the typed bisimulation theories of binary sessions with located queues.

Existing works about expressiveness in process algebra is based on encodings; for example, the early work [19] compares expressiveness of synchronous and asynchronous CCS through the impossibility of an encoding. Our work focus on the semantics of types based on the language acceptance of local traces induced by types without encodings. Another paper [9] compares the expressiveness of several process algebras (asynchronous π, distributed π, ambients) through the use of encodings in order to state possibility and impossibility results. Our approach is different, as we use both operation expressiveness and language comparisons.

The syntax and semantics for interruptible sessions have been defined in [12] and are driven by implementation. The gap in expressiveness created by the addition of operator for exceptional behaviours has been studied in [3]. However, the setting is different and the comparisons are based on Turing-(in)completeness of the different calculi defined, whereas, in § 5.2 we use a comparison based on language inclusion. Our interrupt [12, 8] differs from exceptions in sessions studied in [5, 13, 4] as we provide distributed mechanisms for exceptional behaviours. None of [12, 5, 13, 4, 8] studies the expressiveness.

Acknowledgements. This work has been partially sponsored by: EPSRC EP/K011715/1, EPSRC EP/K034413/1, and EPSRC EP/L00058X/1, EU project FP7-612985 UpScale, EU COST Action IC1201 BETTY and Laboratoire d'Informatique de Paris 6 (UPMC).

References

1 Ocean Observatories Initiative (OOI). `http://www.oceanobservatories.org/`.
2 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. In *FMOODS/FORTE 2013*, pages 50–65, 2013.
3 Mario Bravetti and Gianluigi Zavattaro. On the expressive power of process interruption and compensation. *Mathematical Structures in Computer Science*, 19(3):565–599, 2009.
4 Sara Capecchi, Elena Giachino, and Nobuko Yoshida. Global escape in multiparty sessions. *MSCS*, 29:1–50, 2015.
5 Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured interactional exceptions in session types. In *CONCUR*, volume 5201 of *LNCS*, pages 402–417. Springer, 2008.
6 Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types. In *PPDP 2014*, pages 146–135. ACM Press, 2014.
7 Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *MSCS*, 760:1–65, 2015.
8 Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: Distributed dynamic verification with multiparty session types and python. *FMSD*, pages 1–29, 2015.
9 Daniele Gorla. On the relative expressive power of asynchronous communication primitives. In *FOSSACS 2006*, pages 47–62, 2006.
10 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138, 1998.
11 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
12 Raymond Hu, Rumyana Neykova, Nobuko Yoshida, Romain Demangeon, and Kohei Honda. Practical interruptible conversations – distributed dynamic verification with session types and python. In *RV 2013*, pages 130–148, 2013.
13 Svetlana Jaksic and Luca Padovani. Exception handling for copyless messaging. *Sci. Comput. Program.*, 84:22–51, 2014.

14 Dimitrios Kouzapas, Nobuko Yoshida, Raymond Hu, and Kohei Honda. On asynchronous eventful session semantics. *MSCS*, 2015.

15 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL*, pages 221–232, 2015.

16 Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015.

17 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP'09*, number 5502 in LNCS. Springer, 2009.

18 Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty Session C: Safe parallel programming with message optimisation. In *TOOLS*, volume 7304 of *LNCS*, pages 202–218. Springer, 2012.

19 Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

20 Joachim Parrow. Expressiveness of process algebras. *Electr. Notes Theor. Comput. Sci.*, 209:173–186, 2008.

21 Jorge A. Pérez. *Higher-Order Concurrency: Expressiveness and Decidability Results*. PhD thesis, University of Bologna, 2010.

22 Scribble Project homepage. `www.scribble.org`.

Secure Refinements of Communication Channels

Vincent Cheval[1], Véronique Cortier[2], and Eric le Morvan[2]

1 **School of Computing, University of Kent, UK & LORIA, INRIA, France**
2 **LORIA, CNRS, France**

Abstract

It is a common practice to design a protocol (say Q) assuming some secure channels. Then the secure channels are implemented using any standard protocol, e.g. TLS. In this paper, we study when such a practice is indeed secure.

We provide a characterization of both confidential and authenticated channels. As an application, we study several protocols of the literature including TLS and BAC protocols. Thanks to our result, we can consider a larger number of sessions when analyzing complex protocols resulting from explicit implementation of the secure channels of some more abstract protocol Q.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Protocol, Composition, Formal methods, Channels, Implementation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.575

1 Introduction

When designing a protocol, it is common to assume a secure, confidential, or authentic channel. Authentic channels may be read but not written in. Symmetrically, confidential channels may be written in but not read. Secure channels are both authentic and confidential. For example, payment protocols like 3D-secure are supposed to be run over a secure channel such as TLS. Similarly, many services such as public key registration assume an authenticated channel. How to implement these secure channels is left unspecified and, intuitively, the security of a payment protocol should not depend on the particular choice of implementation of its secure channels. A typical example of a generic realization of a secure channel is TLS. For authentication, one usually relies on a password-based authentication or on previously established keys (used e.g. for signature or MACs). Is it safe to use these protocols in any context? What is a secure or authenticated channel? This paper aims at characterizing channels that have security properties. For example, assume Q is a secure protocol (e.g. a payment protocol) that requires a secure channel. Which properties should a protocol P achieve in order to securely realize the secure channels of Q? These properties should of course be independent of Q since P and Q are typically designed in totally independent contexts. In the remaining of this introduction, Q will refer to the "main" protocol while P will refer to a protocol realizing secure channels (for several notions of security).

Our contributions. Our first contribution is a characterization of both secure, confidential, and authenticated channels. We actually characterize what it means for a channel to be readable or not, and writable or not. Then the realization of a secure channel typically proceeds in two phases. First, some values are established by the protocol P, for example short-term symmetric keys or MAC keys. Quite unsurprisingly, we show that these values need to be secret and appropriately shared. Then the messages of Q are transported or *encapsulated* using the values established by P. For example, the messages of Q may be

encrypted with a key established by P. We provide a characterization of secure encapsulations both for secure, confidential, and authentic channels. A key feature of our characterization is that it is independent of P and Q, which allows for a modular analysis. We show that standard encapsulations (e.g. typical use of encryption, signatures, or MACs) enjoy the requested properties.

Our second and main contribution is to show how to securely compose protocols. Intuitively, our main result guarantees that whenever P is a secure key exchange protocol and $\mathcal{E}$ is a secure encapsulation then $P \cdot^{\mathcal{E}} Q$ is as secure as Q where $P \cdot^{\mathcal{E}} Q$ denotes the protocol obtained from Q by implementing its secure channels using P and $\mathcal{E}$. The interest of our result is twofolds. First, it provides foundational grounds to a common practice where protocols are typically designed and studied independently and then combined. We show that such a practice is actually secure under reasonably light assumptions: primitives shared between P, $\mathcal{E}$, and Q should be tagged as proposed in [4]. Tagging is a standard practice that avoids message confusion. Second, our result provides a technique for analyzing a complex protocol: it is sufficient to analyse its components to deduce security of the whole protocol. To express and prove our result, we have developed a framework, an extension of the applied-pi calculus [2], that allows to easily talk about protocols roles and sessions, a missing aspect in the applied-pi calculus. To illustrate our approach, we show that TLS is a secure implementation of secure channels. Similarly we show that the BAC protocol [1] is also a secure implementation of a secure channel and may be safely used with the Passive Authentication (PA) protocol as prescribed for the biometric passport [1]. Using the CL-Atse tool [18], we analyse several combined protocols. Thanks to our combination result, it is possible to analyse protocols in isolation which allows to consider a larger number of sessions.

Related work. One seminal work on composition is the one of Guttman and Thayer [13]. They show that two protocols can be composed without one damaging the security of the other as soon as they are "independent". However, this independence notion needs to be checked for any protocol execution and cannot be statically checked at the protocol specification level. Later, Guttman [11] provides a criterion on the specification of P and Q such that P can be safely composed with Q. Intuitively, Q should not break some invariant satisfied by P and conversely. While the work of [11] focuses on authentication and secrecy properties, [12] more generally devises a framework for defining protocol goals and designing, step by step, protocols that fulfill them. In [10], the strand space model is used in a modular way, to analyse protocols components by components. The disjunction criteria cannot be checked statically. All these approaches provide a framework that allows to reason modularly when analysing the combination of two protocols P and Q, typically expressing invariants satisfied by P that are shown sufficient to prove security of Q. This simplifies the proof of P combined with Q but requires the knowledge of both protocols. Compared to our work, we propose a criteria for a protocol P to securely implement a secure channel, independently of the protocol Q that will use it (provided primitives are tagged).

Under tagging assumptions similar to ours, it was already shown that P and Q can be safely run in parallel even if they share long-term keys [7]. In passing, we generalize this result to the case where long-term keys may be used as payload. [6] explains when two protocols may be used sequentially, with Q using data established by P. The main difference with our work is that messages may not be transformed when composing protocols. Therefore, [7, 6] cannot be used to (securely) implement abstract channels. Note also that [6] may not consider compromised sessions, that is sessions between honest and dishonest agents. The problem we address here is referred to as *sequential composition* in [16], where the messages of Q are used

as payloads in the composed protocol $P \cdot^{\mathcal{E}} Q$. [16] provides a nice exposition of the generic problem of a protocol Q using a protocol P as subprotocol and lists sufficient (semantical) conditions for combining two protocols. These conditions require again the knowledge of both P and Q. Datta et al. (e.g. [8]) have also studied secure protocol composition in a broader sense: protocols can be composed in parallel, sequentially or protocols may use other protocols as components. However, they do not provide any syntactic conditions for a protocol P to be safely executed in parallel with other protocols. For any protocol P' that might be executed in parallel, they have to prove that the two protocols P and P' satisfy each other invariants. Their approach is thus rather designed for component based design of protocols.

2 Model

Our model is inspired from the applied-pi calculus [2], extended to an explicit notion of roles and agents.

2.1 Messages

Messages are modeled using a typed term algebra. We assume an infinite set of names $\mathcal{N} = \mathcal{N}_D \uplus \mathcal{N}_H$ of *base type* and a set $\mathcal{C}h$ of names of *channel type*. The set $\mathcal{N}_H$ (resp. $\mathcal{N}_D$) represents the names accessible by honest (resp. dishonest) agents. We also a consider an infinite set of variables $\mathcal{X}$ and a finite signature $\mathcal{F}$ of function symbols operating and returning terms of *base type*. More precisely, we consider $\mathcal{F} = \mathcal{F}_c \uplus \mathcal{F}_{cst} \uplus \mathcal{F}_{key}$ where $\mathcal{F}_{cst}$ contains only constants, all functions in $\mathcal{F}_{key}$ are unary, and $\mathcal{F}_c = \{\langle\,\rangle/2, \mathsf{f}_1/n_1, \ldots, \mathsf{f}_k/n_k\}$ contains the binary function symbol $\langle\,\rangle$ used to denote concatenation and other function symbols f_i of arity n_i. Terms are defined as names, variables and function symbols applied to other terms. The set of terms built from $N \subseteq \mathcal{N} \cup \mathcal{C}h$, $X \subseteq \mathcal{X}$ and by applying the function symbols in $F \subseteq \mathcal{F}$ is denoted by $\mathcal{T}(F, N \cup X)$. We denote by $st(t)$ the set of subterms of t. We denote by $vars(t)$ (resp. $names(t)$) the set of variables (resp. names) in t. When $vars(t) = \emptyset$, we say that t is *ground*. To represent events that may occur during a protocol execution, we assume an infinite signature $\mathcal{E}v$ distinct from $\mathcal{F}$. We say that a term $e(t_1, \ldots, t_n)$ with $e \in \mathcal{E}v$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{N} \cup \mathcal{X})$ is an event.

▶ **Example 1.** A standard signature to represent encryption and signature is $\mathcal{F}_{std}$, the signature built from a finite set of constants, functions $\mathcal{F}_{cstd} = \{\mathsf{senc}/2, \mathsf{aenc}/2, \mathsf{sign}/2, \mathsf{h}/1, \langle\rangle/2\}$ and $\mathcal{F}_{kstd} = \{\mathsf{pk}/1, \mathsf{vk}/1\}$. The function symbol senc (resp. aenc) represents the symmetric (resp. asymmetric) encryption. We denote by $\mathsf{pk}(s)$ the public key associated s. The function symbol sign represents the digital signature where $\mathsf{vk}(s)$ is the verification associated to s. We write $\langle u, v\rangle$ as syntactic sugar for $\langle\rangle(u, v)$.

We model the algebraic properties of the cryptographic primitives by a set of inference rules $\mathcal{I}$ composed of composition and decomposition rule described as follows:

$$\frac{x_1 \quad \ldots \quad x_k}{\mathsf{f}(x_1, \ldots, x_k)}\ \mathsf{f}\text{-comp} \qquad \frac{\langle x_1, x_2\rangle}{x_1} \qquad \frac{\langle x_1, x_2\rangle}{x_2}$$

$$\frac{\mathsf{f}(x, u_1, \ldots, u_n) \quad v_1 \quad \ldots \quad v_m}{x}\ \mathsf{f}\text{-decomp}$$

where for all $j \in \{1, \ldots, n\}$, for all $k \in \{1, \ldots, m\}$, $u_j, v_k \in \mathcal{T}(\mathcal{F}_{key}, \mathcal{X})$ and $vars(v_1, \ldots, v_k) \subseteq \{u_1, \ldots, u_n, x\}$. For each $\mathsf{f} \in \mathcal{F}$, the set $\mathcal{I}$ contains a unique f-comp rule and there is no f-decomp rule when $\mathsf{f} \in \mathcal{F}_{key}$. Given a set or sequence of terms S and a term t, the deducibility relation is inductively defined as follows. The term t is deducible from S, denoted

$S \vdash t$, when $t \in S \cup \mathcal{F}_{cst} \cup \mathcal{N}_D$ or there exists a substitution σ and an inference rule in $\mathcal{I}$ with premisses $u_1, \ldots, u_n$ and conclusion u such that $t = u\sigma$ and for all $i \in \{1, \ldots, n\}$, $S \vdash u_i\sigma$.

▶ **Example 2.** Continuing Example 1, we define the set $\mathcal{I}_{std}$ of decomposition rules as follows.

$$\frac{\mathsf{senc}(x,y) \quad y}{x} \qquad \frac{\mathsf{aenc}(x,\mathsf{pk}(y)) \quad y}{x} \qquad \frac{\mathsf{sign}(x,y) \quad \mathsf{vk}(y)}{x} \qquad \frac{\langle x,y\rangle}{x} \qquad \frac{\langle x,y\rangle}{y}$$

We have that $\mathsf{senc}(\langle a,c\rangle,k), k \vdash a$ but $\mathsf{aenc}(\langle a,c\rangle,\mathsf{pk}(k)),\mathsf{pk}(k) \nvdash a$.

2.2 Agents

In standard process algebra (*e.g.* [2]), the notion of agents is usually implicit. Typically, a process that models the behavior of the different honest agents is a single process where all agents are implicitly represented. However, to model protocol composition, we need to explain how to compose each role and thus we need to talk about each agent separately. Therefore, we explicit the presence of agents in our model. Interestingly, our model may also be used to specify semi-honest agents which may directly communicate with the attacker during the protocol execution, still hiding some secrets from him. We consider an infinite set of agents $\mathcal{A}gt = \{A, B, \ldots\} = \mathcal{A}gt_H \uplus \mathcal{A}gt_D$ where $\mathcal{A}gt_H$ and $\mathcal{A}gt_D$ represent respectively honest and dishonest agents. Each agent possesses private data such as keys. Thus, we consider $\mathcal{N}_{\mathcal{A}gt}$ a subset of $\mathcal{N}$ as an infinite partition $\mathcal{N}_{\mathcal{A}gt} = \biguplus_{A \in \mathcal{A}gt} \mathcal{N}_A$ where $\mathcal{N}_A$ intuitively are the names accessible by the agent A. By convention, $k[A]$ denotes a name in $\mathcal{N}_A$.

2.3 Protocols

In the spirit of [2], we model protocols through a process algebra. We represent explicitly confidential, secure, and authenticated channels. Formally, we partition the set of channels into three infinite sets $\mathcal{C}h = \mathcal{C}h_a \uplus \mathcal{C}h_c \uplus \mathcal{C}h_s \uplus \mathcal{C}h_p$ where $\mathcal{C}h_a, \mathcal{C}h_c, \mathcal{C}h_s, \mathcal{C}h_p$ respectively represent the sets of authenticated, confidential, secure and public channels. The syntax of our calculus is as follows:

Roles of agent A

$$R_A, R'_A := \quad 0 \mid \mathsf{out}_A(c,u).R_A \mid \mathsf{in}_A(c,v).R_A \mid \mathsf{new}\ k.R_A \mid \mathsf{event}_A(ev).R_A$$

Channel and agent declarations

$$C, C' := \quad R_A \mid \mathsf{new}_{ta}\ c.C \mid C \mid C'$$

Processes

$$P, Q := \quad C \mid P \mid Q \mid !P \mid \mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub}, \mathcal{K}_{prv}).P$$

where $c \in \mathcal{C}h$, $A \in \mathcal{A}gt$, ta is the tuple of agents in C such that c occurs in their role, k is name, u and v are terms, ev is an event, $\mathcal{K}_{pub}$ and $\mathcal{K}_{prv}$ are sets of ground terms with $names(\mathcal{K}_{pub}) \subseteq \mathcal{N}_A$, $names(\mathcal{K}_{prv}) \subseteq \mathcal{N}_{\mathcal{A}gt}$ and $\mathcal{A} \subseteq \mathcal{A}gt$.

The behavior of an agent A is described in a *role* R_A that consists of a sequence of inputs, outputs, creations of names and emissions of events. The role $\mathsf{out}_A(c,u).R_A$ outputs the term u on the channel c and then behaves like R_A. The role $\mathsf{in}_A(c,v).R_A$ inputs a message from channel c and expects it to be an instance of v. The role $\mathsf{new}\ k.R_A$ generates a fresh name k. Processes express how the roles of different agents are combined. The process $\mathsf{new}_{ta}\ c$ allocates an abstract channel to the agents in ta. The process $P \mid Q$ expresses the parallel execution of P and Q. The process $!P$ represents the replication of P. The process $\mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub}, \mathcal{K}_{prv}).P$ selects a new agent A amongst $\mathcal{A}$. The set $\mathcal{K}_{pub}$ typically indicates the public keys of A while $\mathcal{K}_{prv}$ contains the (secret) long term keys known by A. The variables in a role are uniquely bound by the first input in which they appear. The

$(P \mid \mathsf{out}_A(c,u).R_A, \Phi, \mu, \theta) \to (P \mid R_A, \Phi', \mu', \theta)$ OUT
with $\Phi' = \Phi$ if $c \in \mathcal{C}h_c \cup \mathcal{C}h_s$ else $\Phi' = \Phi \cdot [u]$ and $\mu' = \mathsf{rect}(c,u,\mu)$ if $c \notin \mathcal{C}h_p$ else $\mu' = \mu$

$(P \mid \mathsf{in}_A(c,v).R_A, \Phi, \mu, \theta) \to (P \mid R_A\sigma, \Phi, \mu, \theta)$ IN
if $\exists\sigma$ s.t. $dom(\sigma) = vars(v)$ and either $v\sigma \in c\mu$ or else $c \in \mathcal{C}h_p \cup \mathcal{C}h_c$ and $\Phi \vdash v\sigma$

$(P \mid \mathsf{new}\ k.R_A, \Phi, \mu, \theta) \to (P \mid R_A\{^{k'}/_k\}, \Phi, \mu, \theta)$ NEW-K
with k' fresh in $\mathcal{N}_H$ if $A \in \mathcal{A}gt_H$ else $k' \in \mathcal{N}_D$

$(P \mid \mathsf{new}_{ta}\ c.C[R_{A_1}, \ldots, R_{A_n}], \Phi, \mu, \theta) \to (P \mid [R'_{A_1}, \ldots, R'_{A_n}], \Phi, \mu, \theta')$ NEW-C
$\forall i, R'_{A_i} = R_{A_i}$ if $c \notin ch(R_{A_i})$ else $R'_{A_i} = R_{A_i}\{^{c_{A_i}}/_c\}$ with $c_{A_i} \in \mathcal{C}h_p$ if $ta \cap \mathcal{A}gt_D \neq \emptyset$ else $c_{A_i} \in S \cup \bigcup_{B \in ta} \theta(c,B,ta) \setminus \theta(c,A_i,ta)$ and $S \subseteq \mathcal{C}h_a$ fresh (resp. $\mathcal{C}h_c$, $\mathcal{C}h_s$) if $c \in \mathcal{C}h_a$ (resp. $\mathcal{C}h_c$, $\mathcal{C}h_s$). Moreover, $\theta = \theta'$ if $ta \cap \mathcal{A}gt_D \neq \emptyset$ else $\theta' = \mathsf{recc}(\{(c_A, A)\}_{A \in ta}, ta, c, \theta)$.

$(P \mid !Q, \Phi, \mu, \theta) \to (P \mid !Q \mid Q\rho, \Phi, \mu, \theta)$ REPL
with ρ a fresh renaming of $vars(Q)$

$(P \mid \mathsf{event}_A(ev).R, \Phi, \mu, \theta) \xrightarrow{ev} (P \mid R, \Phi, \mu, \theta)$ EVENT

$(P \mid \mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub}, \mathcal{K}_{prv}).Q, \Phi, \mu, \theta) \to (P \mid Q\sigma, \Phi \cdot S, \mu, \theta)$ AGENT
with $\sigma = \{^{A'}/_A\}$, $A' \notin fa(Q)$, $S = \mathcal{K}_{pub}\sigma$ if $A' \in \mathcal{A} \cap \mathcal{A}gt_H$ else $S = \mathcal{K}_{pub}\sigma \cdot \mathcal{K}_{prv}\sigma$

Figure 1 Semantics of configuration.

channels are bound by the operators new . The agents in a process are also bound by agent creation. In a protocol, we assume that a name or variable is syntactically bound only once. A variable (resp. agent, channel) that is not bound in P is free. We denote by $fa(P)$, $ba(P)$, $fv(P)$, $bv(P)$, $fn(P)$ and $bn(P)$ the sets of free and bound agents, variables and names in P respectively. We say that P is closed when $fv(P) = \emptyset$. Given a process P and an agent A, we denote by $ch_A(P)$ the sets of channels that occur in the roles of A in P.

A role is executable if it only outputs terms that may be deduced from its inputs, the generated values (nonces and keys), and the long-term keys used in the role.

▶ **Definition 3.** Let $R_A = r_1.\ldots.r_n$ be a role of an agent A. We say that R_A is *executable* when for all $i \in \{1,\ldots,n\}$, if $r_i = \mathsf{out}_A(c,u)$ then $names(r_1,\ldots,r_i) \cup S \vdash u$ where $S = \{v \mid j < i \wedge (r_j = \mathsf{in}_A(d,v) \vee r_j = \mathsf{new}\ v)\}$. A process P is executable when all the roles in P are executable.

The state of a protocol during its execution is represented by a *configuration* (P, Φ, μ, θ) where P is a closed process, Φ is a sequence of ground terms representing the knowledge of the attacker, μ is a mapping from channels to sets of terms representing the messages sent over non-public channels and θ is a mapping from triplets of channel, agent, tuple of agents to sets of channels. The semantics is given in Figure 1. The rule OUT indicates that the attacker obtains messages on public or authenticated channels. In this rule, $\mathsf{rect}(c,t,\mu)$ is the mapping μ' where t was recorded as being sent over c. Formally, $\mu'(c') = \mu(c')$ for any $c' \neq c$ and $\mu'(c) = \mu(c) \cup \{t\}$. With rule IN the attacker can inject on c any message that he can deduce from his knowledge when c is a public or confidential channel. He can also relay any message that was previously sent on c. The rule NEW-K generates a fresh name of $\mathcal{N}_H$ or $\mathcal{N}_D$ depending on whether the agent A is honest or not. The rule NEW-C allocates to the role of an agent a channel possibly fresh or that has already been used by other roles in different sessions. In this rule, $\mathsf{recc}(S,ta,c,\theta)$ is the mapping θ in which we record the channels allocated to the agents. Formally, $\theta'(c',A',ta') = \theta(c',A',ta')$ for any $A' \notin ta'$ or

$(c', ta') \neq (c, ta)$, and $\theta'(c, A, ta) = \theta(c, A, ta) \cup \{d\}$ for any $(d, A) \in S$. The rule AGENT selects an agent from $\mathcal{A}$ and adds $\mathcal{K}_{pub}$ to the knowledge of the attacker. Additionally, if the agent is dishonest, the rules adds $\mathcal{K}_{prv}$. When $(P, \Phi, \mu, \theta) \xrightarrow{e_1} \ldots \xrightarrow{e_n} (P', \Phi', \mu', \theta')$, we write $(P, \Phi, \mu, \theta) \xrightarrow{e_1 \cdot \ldots \cdot e_n} (P', \Phi', \mu', \theta')$.

▶ **Example 4.** An electronic passport is a paper passport containing a RFID chip that stores the information printed on the passport. The protocols used to access these private data are specified in the International Civil Aviation Organization standard [1]. Before exchanging any private data, an electronic passport and a reader must establish session keys through a key-exchange protocol, called Basic Access Control (BAC), that prevents eavesdropping on further communication. The BAC protocol relies on two keys ke and km that are printed on the passport and thus can be obtained by the reader through optical scanning. We described below the BAC protocol, between a passport (P) and a reader (R). We assume encrypted messages to be tagged with a. The use of tagging will be explained later on.

$$\mathsf{R} \to \mathsf{P} : \mathsf{challenge}$$
$$\mathsf{P} \to \mathsf{R} : n_\mathsf{P}$$
$$\mathsf{R} \to \mathsf{P} : \langle \mathsf{senc}(\langle \mathsf{a}, n_\mathsf{R}, n_\mathsf{P}, k_\mathsf{R} \rangle, ke), \mathsf{mac}(\langle \mathsf{a}, \mathsf{senc}(\langle \mathsf{a}, n_\mathsf{R}, n_\mathsf{P}, k_\mathsf{R} \rangle, ke) \rangle, km) \rangle$$
$$\mathsf{P} \to \mathsf{R} : \langle \mathsf{senc}(\langle \mathsf{a}, n_\mathsf{P}, n_\mathsf{R}, k_\mathsf{P} \rangle, ke), \mathsf{mac}(\langle \mathsf{a}, \mathsf{senc}(\langle \mathsf{a}, n_\mathsf{P}, n_\mathsf{R}, k_\mathsf{P} \rangle, ke) \rangle, km) \rangle$$

After receiving a challenge command from the reader, the passport generates a fresh name n_P that will be used to verify the authenticity of the messages he will receive later on. Upon receiving n_P, the reader generates two nonces $n_\mathsf{R}, k_\mathsf{R}$ and sends back to the passport all three nonces encrypted with the key k_e and a mac with the key k_m. The nonce n_R has also an authenticity purpose whereas k_R will be the reader's contribution to the session keys. The passport then checks the mac using k_m and the cipher by decrypting it using k_e and verifying the presence of n_P in the plain text. If all verifications succeed, the passport generates a nonce k_P, the passport's contribution to the session keys, and sends it to the reader. At the end of the protocol, both reader and passport know k_R and k_P that they use to generate two session keys $\mathsf{f}_1(k_\mathsf{R}, k_\mathsf{P})$ and $\mathsf{f}_2(k_\mathsf{R}, k_\mathsf{P})$. In our syntax, the roles of the reader (R_R) and of the passport (R_P) can be expressed as follows.

$$\begin{aligned} R_\mathsf{P} = \; & \mathsf{in}_\mathsf{P}(c, \mathsf{challenge}).\mathsf{new}\ n_\mathsf{P}.\mathsf{out}_\mathsf{P}(c, n_\mathsf{P}).\mathsf{in}_\mathsf{P}(c, \langle M, \mathsf{mac}(\langle \mathsf{a}, M \rangle, km[\mathsf{P}]) \rangle). \\ & \mathsf{new}\ k_\mathsf{P}.\mathsf{out}_\mathsf{P}(c, \langle N, \mathsf{mac}(\langle \mathsf{a}, N \rangle, km[\mathsf{P}]) \rangle).0 \end{aligned}$$

$$\begin{aligned} R_\mathsf{R} = \; & \mathsf{out}_\mathsf{R}(c, \mathsf{challenge}).\mathsf{in}_\mathsf{R}(c, z).\mathsf{new}\ k_\mathsf{R}.\mathsf{new}\ n_\mathsf{R}.\mathsf{out}_\mathsf{R}(c, \langle U, \mathsf{mac}(\langle \mathsf{a}, U \rangle, km[\mathsf{P}]) \rangle). \\ & \mathsf{in}_\mathsf{R}(c, \langle V, \mathsf{mac}(\langle \mathsf{a}, V \rangle, km[\mathsf{P}]) \rangle).0 \end{aligned}$$

with $c \in \mathcal{C}h_p$, $M = \mathsf{senc}(\langle \mathsf{a}, x, n_\mathsf{P}, y \rangle, ke[\mathsf{P}])$, $N = \mathsf{senc}(\langle \mathsf{a}, n_\mathsf{P}, x, k_\mathsf{P} \rangle, ke[\mathsf{P}])$, $U = \mathsf{senc}(\langle \mathsf{a}, n_\mathsf{R}, z, k_\mathsf{R} \rangle, ke[\mathsf{P}])$ and $V = \mathsf{senc}(\langle \mathsf{a}, z, n_\mathsf{R}, w \rangle, ke[\mathsf{P}])$. An honest reader communicating with unbounded number of passports, possibly dishonest, can be modeled as the process:

$$BAC = \; \mathsf{ag}(\mathsf{R}, \{\mathsf{R}\}, \emptyset, \emptyset).!\mathsf{ag}(\mathsf{P}, \mathcal{P}, \emptyset, \{ke[\mathsf{P}], km[\mathsf{P}]\}).(R_\mathsf{P} \mid R_\mathsf{R})$$

where $\mathcal{P}$ is an infinite set of agents containing honest and dishonest agents and $\mathsf{R} \notin \mathcal{P}$. The following trace would correspond to the execution of a session with a dishonest passport I and a session of an honest one A both in $\mathcal{P}$.

$$\begin{aligned} (BAC, \emptyset, \emptyset, \emptyset) &\to^* (BAC \mid \mathsf{ag}(\mathsf{P}, \mathcal{P}, \emptyset, \{ke[\mathsf{P}], km[\mathsf{P}]\}).(R_\mathsf{P} \mid R_\mathsf{R}), \emptyset, \emptyset, \emptyset) \\ &\to (BAC \mid R_\mathsf{P}\sigma_A \mid R_\mathsf{R}\sigma_A, \emptyset, \emptyset, \emptyset) \\ &\to^* (BAC \mid R_\mathsf{P}\sigma_A \mid R_\mathsf{R}\sigma_A \mid R_\mathsf{P}\sigma_I \mid R_\mathsf{R}\sigma_I, [ke[\mathsf{I}], km[\mathsf{I}]], \emptyset, \emptyset) \\ &\to (BAC \mid R_\mathsf{P}\sigma_A \mid R_\mathsf{R}\sigma_A \mid R_\mathsf{P}\sigma_I \mid Q, [ke[\mathsf{I}], km[\mathsf{I}], \mathsf{challenge}], \emptyset, \emptyset) \\ &\to^* \ldots \end{aligned}$$

where $\mathsf{P}\sigma_A = \mathsf{A}$, $\mathsf{P}\sigma_I = \mathsf{I}$, $R_R\sigma_I = \mathsf{out}_\mathsf{I}(c, \mathsf{challenge}).Q$ and σ_A, σ_I are fresh renaming of bound variables. By convention, $\mu = \emptyset$ (resp. $\theta = \emptyset$) denotes the mapping that maps any argument to the emptyset: $\mu(c) = \emptyset$ (resp. $\theta(c, A, ta) = \emptyset$) for any c, A, ta.

3 Composition

In the previous section, we have defined an abstract notion of confidential, secure, and authenticated channels. In practice, such channels are realized through cryptographic means. Agents first execute some key establishment protocol in order to generate secret session keys. Then they *encapsulate* the messages supposedly sent over a channel using these session keys. A standard case for secure channels consists in using session keys to encrypt subsequent messages. How to encrypt the message is defined by the *encapsulation*. In Section 3.1, we provide a generic definition of encapsulations and identify properties needed for encapsulations to allow for authentication, confidential, and secure channels. We continue in Section 3.2 by characterizing the composition of a key establishment protocol with a process using abstract channels.

3.1 Encapsulation

For our composition result, we *tag* encapsulations and processes. These tags are used to distinguish the parts of a message that correspond to encapsulations from the ones coming from processes. Formally, a tag is a constant from $\mathcal{F}_{cst}$, hence known to the attacker. Given a set $\mathsf{Tag} \subseteq \mathcal{F}_{cst}$, we say that a term t is a Tag-term when for all $t' \in st(t)$, if $t' = \mathsf{f}(t_1, \ldots, t_n)$ for some $\mathsf{f} \in \mathcal{F}_c \backslash \{\langle\,\rangle\}$ and some terms $t_1, \ldots, t_n$ then $t_1 = \langle \mathsf{a}, u \rangle$ for some term u and $\mathsf{a} \in \mathsf{Tag}$.

▶ **Definition 5.** A Tag-encapsulation is a pair $(\mathcal{E}, \mathsf{F})$ where $\mathcal{E}$ is a Tag-term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\mathsf{F} \subseteq \mathcal{T}(\mathcal{F}_{key}, \mathcal{X})$ such that $vars(\mathcal{E}) = \{x, x_1, \ldots, x_n\}$, $\{\mathcal{E}, x_1, \ldots, x_n\} \vdash x$ and for all $t \in st(\mathcal{E})$,

- if $t = \mathsf{f}(v)$ with $\mathsf{f} \in \mathcal{F}_{key}$ then $v \in \{x_1, \ldots, x_n\} \cup \mathcal{F}_{cst}$
- if $t = \mathsf{f}(w, t_1, \ldots, t_n)$ and there exists a f-decomposition rule with $\mathsf{f}(x, u_1, \ldots, u_n), v_1, \ldots, v_m$ as premises then for all $j \in \{1, \ldots, m\}$, for all $i \in \{1, \ldots, n\}$, $v_j = \mathsf{g}(y)$ and $y \in vars(u_i)$ implies $t_i \in \{x_1, \ldots, x_n\} \cup \mathcal{F}_{cst}$. Intuitively, if a f-decomposition rule may be applied to a subterm of an encapsulation using a non atomic key $\mathsf{g}(t_i)$ then t_i must be a variable or a constant.

We denote x by $\mathsf{t}_{\mathcal{E}}$ and $(x_1, \ldots, x_n)$ by $\mathsf{X}_{\mathcal{E}}$. Given two encapsulations $(\mathcal{E}, \mathsf{F})$ and $(\mathcal{E}', \mathsf{F}')$, we write $\mathcal{E} \sim \mathcal{E}'$ when there exists a renaming ρ such that $\mathcal{E}\rho = \mathcal{E}'$, $\mathsf{F}\rho = \mathsf{F}'$, $\mathsf{t}_{\mathcal{E}}\rho = \mathsf{t}_{\mathcal{E}'}$ and $\mathsf{X}_{\mathcal{E}}\rho = \mathsf{X}_{\mathcal{E}'}$. We denote by $\mathcal{E}(t, t_1, \ldots, t_n)$ the term obtained from $\mathcal{E}$ by substituting x by t and x_i by t_i.

In an encapsulation $(\mathcal{E}, \mathsf{F})$, the variable $\mathsf{t}_{\mathcal{E}}$ will be instantiated by the message sent on the channel implemented by the encapsulation whereas the variables in $\mathsf{X}_{\mathcal{E}}$ will be instantiated by the session keys. Note that $\{\mathcal{E}, x_1, \ldots, x_n\} \vdash x$ indicates that an encapsulated messages may always be retrieved using the session keys. The terms in F represent the public keys that can be used to deduce the term encapsulated or to generate an encapsulation with a new message without revealing the session keys.

▶ **Example 6.** In Example 4, we described how the session keys $\mathsf{f}_1(k_{\mathsf{R}}, k_{\mathsf{P}})$ and $\mathsf{f}_2(k_{\mathsf{R}}, k_{\mathsf{P}})$ are established in the BAC protocol. The ICAO standard states that in any other protocol executed after BAC, the messages exchanged should be of the form $\langle u, \mathsf{mac}(\langle \mathsf{b}, u \rangle, \mathsf{f}_1(k_{\mathsf{R}}, k_{\mathsf{P}}))\rangle$ with $u = \mathsf{senc}(\langle \mathsf{b}, M \rangle, \mathsf{f}_2(k_{\mathsf{R}}, k_{\mathsf{P}}))$ for some data M and tag b. This represents in fact the encapsulation of M with the session keys $\mathsf{f}_1(k_{\mathsf{R}}, k_{\mathsf{P}})$ and $\mathsf{f}_2(k_{\mathsf{R}}, k_{\mathsf{P}})$. In our formalism, the encapsulation is defined as $(\mathcal{E}_{\mathsf{BAC}}, \emptyset)$ where $\mathcal{E}_{\mathsf{BAC}} = \langle t, \mathsf{mac}(\langle \mathsf{b}, t \rangle, x_2)\rangle$ with $t = \mathsf{senc}(\langle \mathsf{b}, x \rangle, x_1)$, $\mathsf{t}_{\mathcal{E}_{\mathsf{BAC}}} = x$ and $\mathsf{X}_{\mathcal{E}_{\mathsf{BAC}}} = (x_1, x_2)$.

We use tags to distinguish the encapsulations from the messages actually sent over the network. However, a process can implement different types of channels using different encapsulations with the same tags. We need to ensure that the security of an encapsulation is not compromised when used with other encapsulations. Therefore, to state the different properties that encapsulations must satisfy, we consider a set of encapsulations and not only a unique one. These conditions are easily met by standard encapsulations.

▶ **Definition 7.** Let $\mathcal{S}_e = \mathcal{S}_a \uplus \mathcal{S}_c \uplus \mathcal{S}_s$ be a set of Tag-encapsulations. We say that $\mathcal{S}_e$ allows authentic, confidential and secure channels if the following properties are satisfied: Let $(\mathcal{E}_1, \mathsf{F}_1), \ldots, (\mathcal{E}_n, \mathsf{F}_n) \in \mathcal{S}_e$. Assume that the variables in $\mathcal{E}_1, \ldots, \mathcal{E}_n$ are disjoint. Let σ be a ground substitution such that $dom(\sigma) = vars(\mathcal{E}_1, \ldots, \mathcal{E}_n)$ and let Φ be a ground frame such that $\mathsf{Tag} \cap st(\sigma, \Phi) = \emptyset$. Let I be the set of $i \in \{1, \ldots, n\}$ such that $\Phi \cdot [\mathcal{E}_k\sigma]_{k=1}^n \vdash \mathsf{t}_{\mathcal{E}_i}\sigma$.

1. For all $i \in \{1, \ldots, n\}$, $\forall u \in \mathcal{T}(\mathcal{F}_{key}, \mathsf{X}_{\mathcal{E}_i}\sigma)$, if $\Phi \cdot [\mathcal{E}_k\sigma]_{k=1}^n \vdash u$ then $\Phi \cdot [\mathsf{t}_{\mathcal{E}_k}\sigma]_{k\in I} \vdash u$.
2. For all $i, i' \in \{1, \ldots, n\}$, $\forall u \in st(\mathcal{E}_i) \setminus \mathcal{X}$, $\forall v \in st(\mathcal{E}_{i'}) \setminus \mathcal{X}$, if u and v are unifiable and $\mathsf{root}(u) \neq \{\langle\,\rangle\}$ then $img(mgu(u, v)) \subset \mathcal{X}$.

Moreover, an encapulation is *authentic*, that is $(\mathcal{E}_i, \mathsf{F}_i) \in \mathcal{S}_a$ if it satisfies the properties **[Can read]** and **[Cannot write]**. An encapulation is *confidential*, that is $(\mathcal{E}_i, \mathsf{F}_i) \in \mathcal{S}_c$ if it satisfies the properties **[Cannot read]** and **[Can write]**. Finally, an encapulation is *secure*, that is $(\mathcal{E}_i, \mathsf{F}_i) \in \mathcal{S}_c$ if it satisfies the properties **[Cannot read]** and **[Cannot write]**.

For all ground substitution σ' such that $\mathsf{Tag} \cap st(\sigma') = \emptyset$, if we denote $J = I - i$ then

3. **[Can read]** $[\mathcal{E}_i] \cdot \mathsf{F}_i \vdash \mathsf{t}_{\mathcal{E}_i}$
4. **[Cannot read]** $\Phi \cdot [\mathcal{E}_k\sigma]_{k=1}^n \vdash \mathsf{t}_{\mathcal{E}_i}\sigma$ implies $\Phi \cdot [\mathsf{t}_{\mathcal{E}_k}\sigma]_{k\in J} \vdash \mathsf{t}_{\mathcal{E}_i}\sigma \vee \exists x \in \mathsf{X}_{\mathcal{E}_i}.\Phi \cdot [\mathsf{t}_{\mathcal{E}_k}\sigma]_{k\in J} \vdash x\sigma$
5. **[Can write]** $\Phi \cdot [\mathcal{E}_k\sigma]_{k=1}^n \vdash \mathcal{E}_i\sigma' \Leftrightarrow \varphi \vee \big(\Phi \cdot [\mathsf{t}_{\mathcal{E}_k}\sigma]_{k\in I} \vdash \mathsf{t}_{\mathcal{E}_i}\sigma' \wedge \Phi \cdot [\mathsf{t}_{\mathcal{E}_k}\sigma]_{k\in I} \vdash \mathsf{F}_i\sigma'\big)$
6. **[Cannot write]** $\Phi \cdot [\mathcal{E}_k\sigma]_{k=1}^n \vdash \mathcal{E}_i\sigma'$ implies either φ or the following property:

$$\exists x \in \mathsf{X}_{\mathcal{E}_i}.\Phi' \vdash x\sigma' \wedge \big((\exists j \in N.\mathsf{t}_{\mathcal{E}_i}\sigma' = \mathsf{t}_{\mathcal{E}_j}\sigma \wedge \mathsf{X}_{\mathcal{E}_i}\sigma' \cap \mathsf{X}_{\mathcal{E}_j}\sigma \neq \emptyset) \vee \Phi' \vdash \mathsf{t}_{\mathcal{E}_i}\sigma')\big)$$

where $\varphi = \exists j \in N.(\mathcal{E}_i \sim \mathcal{E}_j \wedge \mathcal{E}_i\sigma' = \mathcal{E}_j\sigma)$, $N = \{1, \ldots, n\}$ and $\Phi' = \Phi \cdot [\mathsf{t}_{\mathcal{E}_k}\sigma]_{k\in I}$.

The set $\mathcal{S}_a$ (resp. $\mathcal{S}_c$, $\mathcal{S}_s$) represents the sets of encapsulations that can be used to implement authentic (resp. confidential, secure) channels. Property 1 indicates that the session keys or their associated public keys cannot be retrieved directly from an encapsulation. Different encapsulations may use for instance the same encryption scheme. However, Property 2 prevents a part of an encapsulation to be mistaken as session key for another encapsulation. Properties 3 to 6 model the access control of an encapsulation. In particular, the term $\mathsf{t}_{\mathcal{E}}$ of an encapsulation allowing reading access can be derived from the encapsulation $\mathcal{E}$ and its public keys F (Property 3). On the other hand, the term $\mathsf{t}_{\mathcal{E}}$ of an encapsulation not allowing reading access should not be derived from the encapsulation without knowing the session keys $\mathsf{X}_{\mathcal{E}}$ (Property 4). Property 5 indicates that an encapsulation allowing writing access can be deduced only if it was already sent on the network (expressed by formula φ) or by generating it from its public keys F and the term $\mathsf{t}_{\mathcal{E}}$ encapsulated. Lastly, Property 6 models that an encapsulation not allowing writing access cannot be generated by an attacker unless already given or some session keys in $\mathsf{X}_{\mathcal{E}}$ are known. In the latter, Property 6 also states that when the term $\mathsf{t}_{\mathcal{E}}$ is not known to the attacker then he must have extracted it from encapsulations previously received. Most common encapsulations satisfy these properties.

▶ **Theorem 8.** *The following encapsulations are:*
authentic: $\mathcal{E}_{\mathsf{sign}} = \mathsf{sign}(\langle \mathsf{a}_{\mathcal{E}\mathsf{sign}}, x\rangle, x_1)$ *and* $\mathcal{E}_{\mathsf{mac}} = \langle x, \mathsf{h}(\langle \mathsf{a}_{\mathcal{E}\mathsf{mac}}, x, x_1\rangle)\rangle$*;*
confidential: $\mathcal{E}_{\mathsf{aenc}} = \mathsf{aenc}(\langle \mathsf{a}_{\mathcal{E}\mathsf{aenc}}, x\rangle, \mathsf{pk}(x_1))$*;*
secure: $\mathcal{E}_{\mathsf{TLS}} = \mathsf{senc}(\langle \mathsf{a}_{\mathcal{E}\mathsf{TLS}}, x\rangle, x_1)$, $\mathcal{E}_{\mathsf{BAC}} = \langle t, \mathsf{mac}(\langle \mathsf{a}_{\mathcal{E}\mathsf{BAC}}, t\rangle, x_2)\rangle$ *with* $t = \mathsf{senc}(\langle \mathsf{a}_{\mathcal{E}\mathsf{BAC}}, x\rangle, x_1)$, *and* $\mathcal{E}_{\mathsf{signcrypt}} = \mathsf{sign}(\langle \mathsf{a}_{\mathcal{E}\mathsf{signcrypt}}, \mathsf{aenc}(\langle \mathsf{a}_{\mathcal{E}\mathsf{signcrypt}}, x\rangle, \mathsf{pk}(x_1))\rangle, x_2)$.

where $\mathsf{a}_{\mathcal{E}\mathsf{sign}}, \mathsf{a}_{\mathcal{E}\mathsf{mac}}, \mathsf{a}_{\mathcal{E}\mathsf{aenc}}, \mathsf{a}_{\mathcal{E}\mathsf{TLS}}, \mathsf{a}_{\mathcal{E}\mathsf{BAC}}, \mathsf{a}_{\mathcal{E}\mathsf{signcrypt}}$ *are constants.*

Furthermore, the set of encapsulations $\{(\mathcal{E}_{\mathsf{sign}}, \{\mathsf{vk}(x_1)\}),\ (\mathcal{E}_{\mathsf{mac}}, \emptyset),\ (\mathcal{E}_{\mathsf{BAC}}, \emptyset),\ (\mathcal{E}_{\mathsf{TLS}}, \emptyset),\ (\mathcal{E}_{\mathsf{signcrypt}}, \emptyset),\ (\mathcal{E}_{\mathsf{aenc}}, \{\mathsf{pk}(x_1)\})\}$ *allows for authentic, confidential and secure channels.*

In the rest of this paper, we assume the existence of a set of encapsulations $\mathcal{S}_e$ allowing authentic, secure and confidential channels.

3.2 Composition of protocols

Encapsulations use session keys, which are established by a key exchange protocol. To express the requested property of this protocol, we need to annotate it with events that specify which keys are established for which channels and agents.

Considering a context of channel and agent declarations C and a set of channels S, we denote by $C|_{\overline{S}}$ the context C where all $\mathsf{new}_{ta}\ c$ with $c \in S$ are removed. We denote by $T_{\mathcal{A}gt}$ the set of tuples of agents. We consider special events $\mathsf{Ev} = \{\mathsf{ev}_1, \mathsf{ev}_2, \ldots \in \mathcal{E}v\}$.

▶ **Definition 9.** Let $P = C[R_1, \ldots, R_n]$ be a process with C an agent and channel declaration context such that $R_1, \ldots, R_n$ are roles of agents $A_1, \ldots, A_n$ respectively. Let S be a set of channels such that $channels(C) \cap S = \emptyset$. Let ρ be a mapping from S to $T_{\mathcal{A}gt} \times \mathcal{S}_e$. We say that a process $\tilde{P}$ is an annotation of P under ρ if $\tilde{P} = C[R'_1, \ldots, R'_n]$ where for all $i \in \{1, \ldots, n\}$,

$$R'_i = R_i.\mathsf{event}_{A_i}(\mathsf{ev}_i(c_1, ta_1, ts_1, tp_1)).\ldots.\mathsf{event}_{A_i}(\mathsf{ev}_i(c_m, ta_m, ts_m, tp_m))$$

where $\{c_1, \ldots, c_m\} = \{c \in dom(\rho) \mid c\rho = (ta, (\mathcal{E}, \mathsf{F})) \wedge A_i \in st(ta)\}$ and $\forall j \in \{1, \ldots, m\}$, $c_j\rho = (ta_j, (\mathcal{E}, \mathsf{F}))$, $ts_j = (u_1, \ldots, u_{|\mathsf{X}_\mathcal{E}|})$, $tp = \mathsf{F}(u_1, \ldots, u_{|\mathsf{X}_\mathcal{E}|})$ for some $(\mathcal{E}, \mathsf{F})$ and terms $u_1, \ldots, u_{|\mathsf{X}_\mathcal{E}|}$ such that if $c \in \mathcal{C}h_a$ (resp. $\mathcal{C}h_c$, $\mathcal{C}h_s$) then $(\mathcal{E}, \mathsf{F})$ allows authentic (resp. confidential, secure) channels.

At the end of each role R_i, we add the events ev_i for the channels $c_1, \ldots, c_m$ that the agent is supposed to establish. Events $\mathsf{ev}_i(c, ta, ts, tp)$ are composed of four elements: a channel c that the agent wants to instantiate, a tuple of agents ta indicating who is sharing the channel c, a tuple of session keys ts that will be used in the encapsulation $(\mathcal{E}, \mathsf{F})$ to implement c, and lastly a tuple tp of public keys associated to the session keys and F. Typically, we will require that the session keys in ts remain secret for honest agents while their associated public keys in F are made public.

▶ **Example 10.** Continuing Example 4 and thanks to Theorem 8, the encapsulation $(\mathcal{E}_{\mathsf{BAC}}, \emptyset)$ provides the passport and reader with a secure channel, denoted $c_s \in \mathcal{C}h_s$, once BAC has been executed. The fact that BAC is supposed to establish a secure channel for P and R is expressed by the mapping $\rho = \{c_s \to ((\mathsf{P}, \mathsf{R}), (\mathcal{E}, \emptyset))\}$. The corresponding annotation of BAC under ρ is as follows:

$$\begin{aligned} \tilde{BAC} = \quad & C_{BAC}[R_\mathsf{P}.\mathsf{event}_\mathsf{P}(\mathsf{ev}_1(c_s, (\mathsf{P}, \mathsf{R}), (\mathsf{f}_1(y, k_\mathsf{P}), \mathsf{f}_2(y, k_\mathsf{P})))) \\ & \mid R_\mathsf{R}.\mathsf{event}_\mathsf{R}(\mathsf{ev}_2(c_s, (\mathsf{P}, \mathsf{R}), (\mathsf{f}_1(k_\mathsf{R}, w), \mathsf{f}_2(k_\mathsf{R}, w))))] \end{aligned}$$

where $C_{BAC}[\_] = \mathsf{ag}(\mathsf{R}, \{\mathsf{R}\}, \emptyset, \emptyset).!\mathsf{ag}(\mathsf{P}, \mathcal{P}, \emptyset, \{ke[\mathsf{P}], km[\mathsf{P}], data[\mathsf{P}]\}).\_$. Note that the session keys are different and reflect the respective views on the session keys of the passport and the reader.

▶ **Definition 11.** Let C and C' be two channel and agent declaration contexts. We say that C and C' are composable if there exist contexts C_1, C_2, C'_1, C'_2 such that C_1 and C'_1 are sequences of agent declarations with $ba(C_1) \cap ba(C'_1) = \emptyset$, $C = C_1[C_2]$, $C' = C'_1[C'_2]$ and C_2, C'_2 only differ from the content of $\mathcal{K}_{pub}$, $\mathcal{K}_{prv}$ in the instances of $\mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub}, \mathcal{K}_{prv})$.

We define their composition, denoted $\mathsf{C}^{C,C'}$, as the context $C_1[C_1'[C_3]]$ with C_3 being the context C_2 where all instances of $\mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub}, \mathcal{K}_{prv})$ are replaced by $\mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub} \cup \mathcal{K}_{pub}', \mathcal{K}_{prv} \cup \mathcal{K}_{prv}')$ and $\mathsf{ag}(A, \mathcal{A}, \mathcal{K}_{pub}', \mathcal{K}_{prv}')$ is in C_2'.

The composability of the channel and agent declaration contexts ensures that the roles of the process Q can be sequentially composed with the roles of the process P. For instance, they should have similar replications, agent declarations or even channel declarations. However, we do not require that an agent in P and Q to have the same private ($\mathcal{K}_{prv}$) or public ($\mathcal{K}_{pub}$) data. We also allow an agent to be declared in one context but not in the other one if declared upfront.

▶ **Example 12.** One of the protocols that are executed after BAC is the Passive Authentication protocol which provides an authentication mechanism proving that the content of the RFID chip is authentic. In fact the ICAO standard also indicates that the chip must contain a signature by the Document Signer authority (D) of a hash of the private data $data[\mathsf{P}]$, $sod \stackrel{def}{=} \mathsf{sign}(\langle \mathsf{a}, \mathsf{h}(\langle \mathsf{a}, data[\mathsf{P}]\rangle)\rangle, sk[\mathsf{D}])$. During the Passive Authentication protocol, after receiving on the secure channel a challenge from the reader, the passport sends back this signature that is checked by the reader.

$\mathsf{R} \rightarrow_{sec} \mathsf{P} : \mathsf{read}$
$\mathsf{P} \rightarrow_{sec} \mathsf{R} : \langle data, \mathsf{sign}(\langle \mathsf{a}, \mathsf{h}(\langle \mathsf{a}, data\rangle)\rangle, sk)$

where sk is the signing key of the Document Signer authority. In our calculus, the roles of the reader (Q_{R}) and of the passport (Q_{P}) can be described as follows:

$Q_{\mathsf{P}} = \mathsf{in}_{\mathsf{P}}(c_s, \mathsf{read}).\mathsf{out}_{\mathsf{P}}(c_s, \langle data[\mathsf{P}], sod\rangle)$
$Q_{\mathsf{R}} = \mathsf{out}_{\mathsf{R}}(c_s, \mathsf{read}).\mathsf{in}_{\mathsf{R}}(c_s, \langle x', \mathsf{sign}(\langle \mathsf{a}, \mathsf{h}(\langle \mathsf{a}, x'\rangle)\rangle, sk[\mathsf{D}])\rangle)$

The complete representation of the system is given by $PA = C_{PA}[\mathsf{new}_{(R,P)}\ c_s.(Q_P \mid Q_R)]$ where C_{PA} is the following context:

$$C_{PA} = \mathsf{ag}(\mathsf{D}, \{\mathsf{D}\}, \{\mathsf{vk}(sk[\mathsf{D}])\}, \{sk[\mathsf{D}]\}).\mathsf{ag}(\mathsf{R}, \{\mathsf{R}\}, \emptyset, \emptyset).!\mathsf{ag}(\mathsf{P}, \mathcal{P}, \emptyset, \{data[\mathsf{P}]\}).\_$$

Continuing Example 10, C_{PA} and C_{BAC} are composable and $\mathsf{C}^{C_{PA},C_{BAC}}$ is the context:

$$\mathsf{ag}(\mathsf{D}, \{\mathsf{D}\}, \{\mathsf{vk}(sk[\mathsf{D}])\}, \{sk[\mathsf{D}]\}).\mathsf{ag}(\mathsf{R}, \{\mathsf{R}\}, \emptyset, \emptyset).!\mathsf{ag}(\mathsf{P}, \mathcal{P}, \emptyset, \{ke[\mathsf{P}], km[\mathsf{P}], data[\mathsf{P}]\}).\_$$

Let S be a set of channels. Let ρ be a mapping from S to $T_{\mathcal{A}gt} \times \mathcal{S}_e$. We say that two processes P and Q are *composable under* ρ if $P = C[R_1, \ldots, R_n]$, $Q = C'[R_1', \ldots, R_n']$ where R_i, R_i' are roles of the same agent A_i for $i = 1 \ldots n$, C and $C'|_{\overline{S}}$ are composable and for all $c \in dom(\rho)$, if $c\rho = (ta, (\mathcal{E}, \mathsf{F}))$ then for all $i \in \{1, \ldots, n\}$, $c \in ch_{A_i}(Q)$ is equivalent to $A_i \in ta$. This reflects the fact that agents using channel c should be explicitly listed as authorized agents for c.

The composability between P and Q ensures that the agents in Q sharing abstract authentic, confidential and secure channels are correctly represented in ρ.

▶ **Definition 13.** Let S be a set of channels. Let ρ be a mapping from S to $T_{\mathcal{A}gt} \times \mathcal{S}_e$. Let $P = C[R_1, \ldots, R_n]$ and $Q = C'[R_1', \ldots, R_n']$ two closed composable processes under ρ.

For all $\tilde{P} = C[\tilde{R}_1, \ldots, \tilde{R}_n]$ annotations of P under ρ, the implementation of Q by $\tilde{P}$ through ρ, denoted $\tilde{P} \cdot^{\rho} Q$, is the process $C_0[R_1.R_1'', \ldots, R_n.R_n'']$ where $C_0 = \mathsf{C}^{C,C'|_{\overline{S}}}$ and for all $i \in \{1, \ldots, n\}$, R_i'' is defined as R_i' where all instances of $\mathsf{out}_A(c, u)$ (resp. $\mathsf{in}_A(c, u)$) are replaced by $\mathsf{out}_A(c_{pub}, \mathcal{E}\sigma)$ (resp. $\mathsf{in}_A(c_{pub}, \mathcal{E}\sigma)$) when $c\rho = (ta, (\mathcal{E}, \mathsf{F}))$, $\mathsf{t}_{\mathcal{E}}\sigma = u$ and $\mathsf{event}_A(\mathsf{ev}_i(c, ta, \mathsf{X}_{\mathcal{E}}\sigma, \mathsf{F}\sigma))$ is in $\tilde{R}_i$ for some substitution σ.

▶ **Example 14.** Continuing Example 12, the implementation of PA by $B\tilde{A}C$ through ρ is thus the process $B\tilde{A}C \cdot^{\rho} PA = \mathsf{C}^{C_{PA},C_{BAC}}[R_P.Q_P' \mid R_R.Q_R']$ where Q_P' and Q_R' are defined

as follows:

$$Q'_{\mathsf{P}} = \mathsf{in}_{\mathsf{P}}(c_{pub}, \mathcal{E}_{\mathsf{BAC}}(\mathsf{read}, K_1, K_2)).\mathsf{out}_{\mathsf{P}}(c_{pub}, \mathcal{E}_{\mathsf{BAC}}(\langle data[\mathsf{P}], sod\rangle, K_1, K_2))$$
$$Q'_{\mathsf{R}} = \mathsf{out}_{\mathsf{R}}(c_{pub}, \mathcal{E}_{\mathsf{BAC}}(\mathsf{read}, K'_1, K'_2)).\mathsf{in}_{\mathsf{R}}(c_{pub}, \mathcal{E}_{\mathsf{BAC}}(\langle x, \mathsf{sign}(\langle \mathsf{a}, \mathsf{h}(\langle \mathsf{a}, x\rangle)\rangle, sk[\mathsf{D}])\rangle, K'_1, K'_2))$$

with $K_1 = \mathsf{f}_1(y, k_{\mathsf{P}})$, $K_2 = \mathsf{f}_2(y, k_{\mathsf{P}})$, $K'_1 = \mathsf{f}_1(k_{\mathsf{R}}, w)$, $K'_2 = \mathsf{f}_2(k_{\mathsf{P}}, w)$. Note that the ICAO standard describes in fact the Passive Authentication protocol as the process $C[Q'_P \mid Q'_R]$ (without tags). With our result, we may study the simpler process $C[\mathsf{new}_{(P,R)}\ c_s.(Q_P \mid Q_R)]$.

4 Security property

It is easy to state secrecy in our formalism, using a special event $\mathsf{Sec} \in \mathcal{E}v$: any term occurring in a Sec event should remain secret unless the corresponding session involves a dishonest agent.

▶ **Definition 15.** Let Q be closed process containing contains some events of the form $\mathsf{Sec}(t, (A_1, \ldots, A_n))$ where t is a term and $A_1, \ldots, A_n$ are some agents. Let Φ be a closed frame. We say that Q preserves secrecy if for all $(Q, \emptyset, \emptyset, \emptyset) \xRightarrow{ev_1 \cdot \ldots \cdot ev_m} (Q', \Phi', \mu', \theta')$, for all $i \in \{1, \ldots, n\}$, if $ev_i = \mathsf{Sec}(t', (A'_1, \ldots, A'_n))$ for some t' and some honest agents $A'_1, \ldots, A'_n$ then $\Phi' \not\vdash t'$.

We may also specify the properties requested from a key exchange protocol P: P should preserve the secrecy of the session keys occurring in its events and should ensure that the associated public keys are public. Moreover, P also needs to ensure that a session key cannot be used to implement two different channels and that honest agents sharing a channel will share the same session keys for this channel. In such a case, we say that P is a *secure channel establishment protocol*.

▶ **Definition 16.** Let $P = C[R_1, \ldots, R_n]$ be a closed process. Let $\tilde{P}$ be an annotation of P under some mapping ρ. We say that $\tilde{P}$ is a *secure channel establishment protocol* when for all $(\tilde{P}, \emptyset, \emptyset, \emptyset) \xRightarrow{e_1 \cdot \ldots \cdot e_m} (P', \Phi', \mu', \theta')$, for all $i \in \{1, \ldots, m\}$, if $e_i = ev(c, ta, (s_1, \ldots, s_\ell), (u_1, \ldots, u_q))$ such that $ev \in \mathsf{Ev}$, all agents in ta are honest then for all $k \in \{1, \ldots, \ell\}$, $\Phi' \not\vdash s_k$ and for all $k \in \{1, \ldots, q\}$, $\Phi' \vdash u_k$. Moreover, for all $j \in \{1, \ldots, m\}$, if $ev_j = ev'(c', ta', (s'_1, \ldots, s'_{\ell'}), (u'_1, \ldots, u'_{q'}))$ for some $ev' \in \mathsf{Ev}$, some channel c', some tuple ta' of agents and some tuples $(s'_1, \ldots, s'_{\ell'})$ and $(u'_1, \ldots, u'_{q'})$ of terms then

- either $ta \neq ta'$ or $c \neq c'$ or $ev = ev'$ implies $\forall k \in \{1, \ldots, \ell\}, \forall k' \in \{1, \ldots, \ell'\}$, $s_k \neq s'_{k'}$
- or one of the two following properties is satisfied :
  - $(s_1, \ldots, s_\ell) = (s'_1, \ldots, s'_{\ell'})$ and $(u_1, \ldots, u_q) = (u'_1, \ldots, u'_{q'})$.
  - $\forall k \in \{1, \ldots, \ell\}, \forall k' \in \{1, \ldots, \ell'\}$, $s_k \neq s'_{k'}$.

The first item indicates that the session keys used for a channel between some honest agents are necessarily different from session keys used for a different channel between any kind of agents, whether they are honest, dishonest or a mix of both. The second item requires that for matching channels and sets of agents, either the session keys perfectly match or they are all different.

We are now ready to state our main result: if P is a secure channel establishment protocol and if Q preserves secrecy using some secure, confidential, or authentic channels, then Q may safely use P to implement its channels. The proof of Theorem 17 is available in a companion report [5].

▶ **Theorem 17.** *Let* tag_A *and* tag_B *be two disjoint sets of tags. Let* $\mathcal{S}_e$ *be a set of* tag_A*-encapsulation allowing authentic, confidential, and secure channels. Let* ρ *be a mapping from channels to* $T_{\mathcal{A}gt} \times \mathcal{S}_e$*. Let* P *and* Q *be two closed executable composable* tag_B*-processes under* ρ *such that* P *and* Q *do not share names and* $fa(P) = fa(Q) = \emptyset$*. Let* $\tilde{P}$ *be an annotation of* P *under* ρ*. If* $\tilde{P}$ *is secure and* Q *preserves secrecy then* $\tilde{P} \cdot^{\rho} Q$ *preserves secrecy as well.*

For simplicity, we prove secure composition w.r.t. secrecy properties but we believe that our result could be easily extended to trace properties.

Sketch of proof. The proof first relies on that fact that the reachability properties are preserved by disjoint parallel composition. In particular, the process $\tilde{P} \mid Q$ is a secure channel establishment protocol and preserves secrecy. The rest of the proof consists in showing that any trace of $\tilde{P} \cdot^{\rho} Q$ is also a trace of $\tilde{P} \mid Q$ with a frame that induces a similar attacker knowledge. More specifically, properties from Definition 7 ensure that tag_B-terms generated by the attacker or obtained from the encapsulations in $\tilde{P} \cdot^{\rho} Q$ do not give any relevant knowledge to the attacker and can be replaced by fresh names. This allows us to obtain a trace without tag_B-terms and so without encapsulations. Lastly, since $\tilde{P} \mid Q$ is a secure channel establishment protocol, we can always match two encapsulations having same session keys with the corresponding abstract channel in $\tilde{P} \mid Q$. ◀

▶ **Example 18.** Continuing Example 14, the annotation under ρ of the Basic Access Control $\tilde{BAC}$ is secure and the Passive Authentication $C_{PA}[\mathsf{new}\ c_s.(Q_P.\mathsf{event}_{\mathsf{P}}(\mathsf{Sec}(data[\mathsf{P}], (\mathsf{P}, \mathsf{R}))) \mid Q_R)]$ preserves secrecy (of the private data). Hence, thanks to Theorems 8 and 17, the implementation of PA by $\tilde{BAC}$ through ρ, $\mathsf{C}^{C_{PA}, C_{BAC}}[R_P.Q'_P.\mathsf{event}_{\mathsf{P}}(\mathsf{Sec}(data[\mathsf{P}], (\mathsf{P}, \mathsf{R}))) \mid R_R.Q'_R]$, preserves secrecy.

5 Case studies

We show that our approach can be applied to deployed protocols such as the biometric passport or TLS applied to 3D-secure. As an application, we show that the automatic analysis through the CL-Atse tool can be significantly speed up when the number of sessions goes higher.

5.1 Biometric passport

Our running example is the combination of the Basic Access Control (BAC) protocol with the Passive Authentication (PA) protocol from the electronic passports. Actually, PA is not the only protocol executed after BAC. Another authentication mechanism is used to prevent cloning of the passport chip. This protocol, called *Active Authentication protocol* (AA), also uses the same session keys and encapsulations than PA. Using the CL-Atse tool [18], we show for different scenarios that BAC is a secure channel establishment protocol and that PA and AA both preserve secrecy. Thanks to our main result, this yields security of the combined protocol, where BAC implements the secure channel of PA and AA. For comparaison purpose, we also analyze directly the combined protocol with CL-Atse. These analysis are reported in Section 5.3

5.2 TLS and 3D-secure

Our results also apply to other complex systems. We study the *Visa 3D-secure protocol* [17] used by several websites for internet banking and that relies on secure channels implemented

by the well known TLS protocol. The Visa 3D secure protocol is an authenticated payment method between a card holder and a merchant during an electronic payment. This protocol aims to ensure authentication of the card holder as well as confirmation that the card holder is authorized by his bank to make the payment. Lastly, the protocol also aims to ensure the secrecy of the card holder's banking information, the payment amount and other data.

The protocol involves four types of participants: a card holder (C), a merchant (M), a centralized structure called Visa Directory Servers (DS) and the card issuer's servers called Access Control Servers (ACS). The main role of the Visa Directory Servers is to transfer card holder's information between the Access Control Servers and the merchant.In itself, the 3D secure protocol is already a complex protocol with multiple exchanges of messages. But the protocol also requires most messages to be exchanged trough a TLS channel. More specifically, messages of the 3D secure protocol shall be encrypted with a symmetric session key previously established with TLS. In our model, this means that the messages are encapsulated by $(\mathcal{E}_{\mathsf{TLS}}, \emptyset)$, as defined in Theorem 8.

The well known TLS protocol [15, 9] aims at establishing a secure channel between a client and a server. Using the CL-Atse tool, we show that TLS (Basic TLS handshake, in the RSA mode) is indeed a secure channel establishment protocol.

Note that for one session of the Visa 3D secure protocol yields four sessions of the TLS protocol: one channel between C and M, between C and ACS, between ACS and DS and finally between M and DS. This renders the verification of even one session of 3D secure protocol with the channels implemented by TLS a complex task (more than thirty five messages exchanged per session).

5.3 Analysis with CL-Atse

We applied the automatic verification tool CL-Atse [18] on a Dell T1700 computer (16 Go RAM, 3.40 GHz CPU). The corresponding time of analysis are displayed below.

		Computation time (in seconds, timeout set to 24 hours)							
protocols		TLS & 3D secure		BAC & PA		BAC & AA		BAC & PA & AA	
complete system (C) or separated analysis (S)		S	C	S	C	S	C	S	C
number of sessions considered	1	0.2	0.1	0.7	0.1	0.7	0.1	0.7	0.2
	2	1350	time out	6.2	1.6	6.2	1.6	6.5	43156
	3	time out	time out	9133	time out	9133	time out	9185	time out

Amongst the tools able to verify security protocols for a bounded number of sessions, CL-Atse is well known and considered to be one of the fastest. However, in the case of the 3D-secure protocol, the tool already fails to verify one session with all channels implemented as we reached a time out set to 24 hours of computation. Thus, to obtain meaningful results with the 3D-secure protocol, we considered the case where only the channel between the card holder and the merchant is implemented. Already in this case, we can see a clear benefit from analyzing separately 3D-secure and TLS when considering two sessions. Indeed, the verification can be performed under 25 minutes when analysing the protocols separately whereas the tool was reaching a time out when considering the complete system. We obtain similar results with the Basic Access Control protocol, the Active Authentication protocol and the Passive Authentication protocol. Note that for verification tools handling unbounded number of sessions (*e.g.* ProVerif [3], Tamarin [14]), the gain in time would probably be less significant since these tools do not systematically explore all interleavings.

6 Conclusion

We have shown how to securely compose a protocol with the implementation of its channels. We have provided a characterization for the three most common types of channels: secure, confidential, and authentic channels. We plan to consider other types of communication channels like anonymous channels. This will certainly require to extend our approach to equivalence properties.

Our composition result holds for a class of primitives that encompasses all standard cryptographic primitives. We plan to extend it to a larger class of primitives, including in particular exclusive or or homomorphic encryption.

Our result assumes a light tagging of the primitives, to ensure that an encapsulation cannot be confused with a message coming from the protocols. While tagging is reasonable, it is not often done in practice. On the other hand standard protocols typically enjoy some non unifiability properties that prevent such confusion. We believe that our result could be extended to a general notion of non unifiability of the terms, without having to require explicit tagging.

Acknowledgments. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 258865, project ProSecure.

References

1 Machine readable travel document. Technical Report 9303, International Civil Aviation Organization, 2008.

2 M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

3 Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. CSFW'01*, 2001.

4 Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In Andrew Gordon, editor, *Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*, April 2003.

5 V. Cheval, V. Cortier, and E. Le-Morvan. Secure refinements of communication channels. Research report RR-8790, Inria, 2015.

6 Ştefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 322–336, Edinburgh, Scotland, UK, July 2010. IEEE Computer Society Press.

7 Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, February 2009.

8 Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electr. Notes Theoretical Computer Science*, 172:311–358, 2007.

9 T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2 (rfc 5246). Technical report, IETF, 2008.

10 Thomas Gibson-Robinson, Allaa Kamil, and Gavin Lowe. Verifying layered security protocols. *Journal of Computer Security*, 23(3), 2015.

11 Joshua D. Guttman. Authentication tests and disjoint encryption: a design method for security protocols. *Journal of Computer Security*, 12(3–4):409–433, 2004.

12 Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):203–267, 2004.

13 Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proc. 13th Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Comp. Soc. Press, 2000.

14 Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

15 Christopher Meyer and Jorg Schwenk. Lessons learned from previous ssl/tls attacks : A brief chronology of attacks and weaknesses. In *IACR Cryptology ePrint*, 2013.

16 Sebastian Moedersheim and Luca Viganò. Sufficient conditions for vertical composition of security protocols. In *ASIACCS*, pages 435–446, 2014.

17 Vijayakrishnan Pasupathinathan, Josef Pieprzyk, Huaxiong Wang, and Joo Yeon Cho. Formal analysis of card-based payment systems in mobile services. In *Fourth Australian information security workshop, conferences in research and practise in information security*, pages 213–220, 2006.

18 Mathieu Turuani. The CL-Atse Protocol Analyser. In *Term Rewriting and Applications – Proc. of RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286, Seattle, WA, USA, 2006.

Failure-aware Runtime Verification of Distributed Systems

David Basin[1], Felix Klaedtke[2], and Eugen Zălinescu[1]

1 **ETH Zürich, Department of Computer Science, Zürich, Switzlerland**
2 **NEC Labs Europe, Heidelberg, Germany**

Abstract

Prior runtime-verification approaches for distributed systems are limited as they do not account for network failures and they assume that system messages are received in the order they are sent. To overcome these limitations, we present an online algorithm for verifying observed system behavior at runtime with respect to specifications written in the real-time logic MTL that efficiently handles out-of-order message deliveries and operates in the presence of failures. Our algorithm uses a three-valued semantics for MTL, where the third truth value models knowledge gaps, and it resolves knowledge gaps as it propagates Boolean values through the formula structure. We establish the algorithm's soundness and provide completeness guarantees. We also show that it supports distributed system monitoring, where multiple monitors cooperate and exchange their observations and conclusions.

1998 ACM Subject Classification C.2.4 Distributed Systems, D.2.4 Software/Program Verification, D.2.5 Testing and Debugging, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic

Keywords and phrases Runtime verification, monitoring algorithm, real-time logics, multi-valued semantics, distributed systems, asynchronous communication

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.590

1 Introduction

Distributed systems are omnipresent and complex, and they can malfunction for many reasons, including software bugs and hardware or network failures. Runtime monitoring is an attractive option for verifying at runtime whether a system behavior is correct with respect to a given specification. But distribution opens new challenges. The monitor itself becomes a component of the (extended) system and like any other system component it may exhibit delays, finite or even infinite, when communicating with other components. Moreover, the question arises whether monitoring itself can be distributed, thereby increasing its efficiency and eliminating single points of failure. Distribution also offers the possibility of moving the monitors close to or integrating them in system components, where they can more efficiently observe local system behavior.

Various runtime-verification approaches exist for different kinds of distributed systems and specification languages [19, 3, 8, 15]. These approaches are of limited use for monitoring distributed systems where components might crash or network failures can occur, for example, when a component is temporarily unreachable and a monitor therefore cannot learn the component's behavior during this time period. Even in the absence of failures, monitors can receive messages about the system behavior in any order due to network delays. A naive solution for coping with out-of-order message delivery is to have the monitor buffer messages and reorder them before processing them. However, this can delay reporting a violation when

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 590–603

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the violation is already detectable on some of the buffered messages. Another limitation concerns the expressivity of the specification languages used by these monitoring approaches. It is not possible to express real-time constraints, which are common requirements for distributed systems. Such constraints specify, for example, deadlines to be met.

In this paper, we present a monitoring algorithm for the real-time logic MTL [11, 1] that overcomes these limitations. Our algorithm accounts for out-of-order message deliveries and soundly operates in the presence of failures that, for example, cause components to crash. In the absence of failures, we also provide completeness guarantees, meaning that a monitor eventually reports the violation or the satisfaction of the given specification. Furthermore, our algorithm allows one to distributively monitor a system. To achieve this, the system is extended with monitoring components that receive observations from system components about the system behavior and the monitors cooperate and exchange their conclusions.

Our monitoring algorithm builds upon a timed model for distributed systems [7]. The system components use their local clocks to timestamp observations, which they send to the monitors. The monitors use these timestamps to determine the elapsed time between observations, e.g., to check whether real-time constraints are met. Furthermore, the timestamps totally order the observations. This is in contrast to a time-free model [9], where the events of a distributed system can only be partially ordered, e.g., by using Lamport timestamps [12]. However, since the accuracy of existing clocks is limited, the monitors' conclusions might only be valid for the provided timestamps. See Section 5 where we elaborate on this point.

We base our monitoring algorithm on a three-valued semantics for the real-time logic MTL, where the interpretation of the third truth value, denoted by $\bot$, follows Kleene logic [10]. For example, a monitor might not know the Boolean value of a proposition at a time point because a message from a system component about the proposition's truth value is delayed, or never sent or received. In this case, the monitor assigns the proposition the truth value $\bot$, indicating that the monitor has a knowledge gap about the system behavior. The truth value $\bot$ is also used by monitors to avoid issuing incorrect verdicts about the system behavior: a monitor only reports the satisfaction of the specification or its negation, and this verdict remains valid no matter how the monitor's knowledge gaps are later resolved when receiving more information about the system behavior. No verdict is output if under the current knowledge the specification evaluates to $\bot$.

To efficiently resolve knowledge gaps and to compute verdicts, each monitor maintains a data structure that stores the parts of the specification—a subformula and an associated time point—that have not yet been assigned a Boolean value. Intuitively speaking, the truth value of the subformula at the given time is $\bot$ under the monitor's current knowledge. These parts are nodes of an AND-OR-graph, where the edges express constraints for assigning a Boolean value to a node. When a monitor receives additional information about the system behavior, it updates its graph structure by adding and deleting nodes and edges, based on the message received. To compute verdicts, the monitors also propagate Boolean values between nodes when possible.

Our main contribution is a novel monitoring algorithm for MTL specifications. It is the first algorithm that efficiently handles observations that can arrive at the monitor in any order. This feature is essential for our approach to monitoring distributed systems, which is our second contribution. Our approach overcomes the limitations of prior runtime-verification approaches for distributed systems. Namely, it handles message delays, it allows one to distribute the monitoring process across multiple system components, and it soundly accounts for failures such as crashes of system components.

We proceed as follows. In Section 2, we introduce the real-time logic MTL with a three-valued semantics. In Section 3, we describe the system assumptions and the requirements.

Table 1 Truth tables for three-valued logical operators (strong Kleene logic [10]).

$\neg$	
t	f
f	t
$\bot$	$\bot$

$\vee$	t	f	$\bot$
t	t	t	t
f	t	f	$\bot$
$\bot$	t	$\bot$	$\bot$

$\wedge$	t	f	$\bot$
t	t	f	$\bot$
f	f	f	f
$\bot$	$\bot$	f	$\bot$

$\rightarrow$	t	f	$\bot$
t	t	f	$\bot$
f	t	t	t
$\bot$	t	$\bot$	$\bot$

In Section 4, we present our monitoring algorithm. In Section 5, we consider the impact of the accuracy of timestamps for ordering observations. In Section 6, we discuss related work. Finally, in Section 7, we draw conclusions. Details, omitted due to space restrictions, can be found in the full version of this paper, which is available from the authors or their webpages.

2 Three-Valued Metric Temporal Logic

For Σ an alphabet, we work with words that are finite or infinite sequences of tuples in $\Sigma \times \mathbb{Q}_+$, where $\mathbb{Q}_+$ is the set of positive rational numbers. We write $|w| \in \mathbb{N} \cup \{\infty\}$ to denote the length of w and (σ_i, τ_i) for the tuple at position i. A *timed word* w is a word where:

(i) $\tau_{i-1} < \tau_i$, for all $i \in \mathbb{N}$ with $0 < i < |w|$, and
(ii) If $|w| = \infty$ then for every $t \in \mathbb{Q}_+$, there is some $i \in \mathbb{N}$ such that $\tau_i > t$.

Observe that (1) requires that the sequence of the τ_is is strictly increasing rather than requiring only that the τ_is increase monotonically, as, e.g., in [1]. This means that there are no fictitious clocks that order tuples with equal τ_is. Instead, it is assumed that everything at time τ_i happens simultaneously and the τ_is already totally order the tuples that occur in w.

We denote the set of infinite timed words over the alphabet Σ by $TW^\omega(\Sigma)$. We often write a timed word $w \in TW^\omega(\Sigma)$ as $(\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$. We call the τ_is *timestamps* and the indices of the elements in the sequence *time points*. For $\tau \in \mathbb{Q}_+$, let $\mathrm{tp}(w, \tau)$ be w's time point i with $\tau_i = \tau$ if it exists. Otherwise, $\mathrm{tp}(w, \tau)$ is undefined.

Let $\mathsf{3}$ be the set $\{\mathsf{t}, \mathsf{f}, \bot\}$, where t (true) and f (false) denote the Boolean values, and $\bot$ denotes the truth value "unknown." Table 1 shows the truth tables of some standard operators over $\mathsf{3}$. Observe that these operators coincide with the Boolean ones when restricted to the set $\mathsf{2} := \{\mathsf{t}, \mathsf{f}\}$ of Boolean values.

We partially order the elements in $\mathsf{3}$ by their knowledge: $\bot \prec \mathsf{t}$ and $\bot \prec \mathsf{f}$, and t and f are incomparable as they carry the same amount of knowledge. Note that $(\mathsf{3}, \prec)$ is a lower semilattice, where $\curlywedge$ denotes the meet.

Throughout the paper, let P be a set of atomic propositions. We extend the partial order $\prec$ over $\mathsf{3}$ to timed words over the alphabet $\Sigma := \mathsf{3}^P$, where X^Y is the set of functions with domain Y and range X. Let $v, v' \in TW^\omega(\Sigma)$, where (σ_i, τ_i) and (σ'_i, τ'_i) are the tuples at position i in v and v', respectively. We define $v \preceq v'$ if $|v| = |v'|$, $\tau_i = \tau'_i$, and $\sigma_i(p) \preceq \sigma'_i(p)$, for every i with $0 \leq i < |v|$ and every $p \in P$. Intuitively, some of the knowledge gaps about the propositions' truth values in v are resolved in v'.

The syntax of the real-time logic MTL is given by the grammar: $\varphi ::= \mathsf{t} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\, \mathsf{S}_I\, \varphi \mid \varphi\, \mathsf{U}_I\, \varphi$, where p ranges over P's elements and I ranges over intervals over $\mathbb{Q}_+$. For brevity, we omit the temporal connectives for "previous" and "next." MTL's three-valued

semantics is defined as follows. Let $i \in \mathbb{N}$ and $w \in TW^\omega(\Sigma)$ with $w = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$.

$$\begin{aligned}
[\![w, i \models \mathsf{t}]\!] &:= \mathsf{t} \\
[\![w, i \models p]\!] &:= \sigma_i(p) \\
[\![w, i \models \neg\varphi]\!] &:= \neg[\![w, i \models \varphi]\!] \\
[\![w, i \models \varphi \vee \psi]\!] &:= [\![w, i \models \varphi]\!] \vee [\![w, i \models \psi]\!] \\
[\![w, i \models \varphi \,\mathsf{S}_I\, \psi]\!] &:= \bigvee_{j \in \{\ell \in \mathbb{N} \mid \tau_i - \tau_\ell \in I\}} \big([\![w, j \models \psi]\!] \wedge \textstyle\bigwedge_{j < k \leq i} [\![w, k \models \varphi]\!]\big) \\
[\![w, i \models \varphi \,\mathsf{U}_I\, \psi]\!] &:= \bigvee_{j \in \{\ell \in \mathbb{N} \mid \tau_\ell - \tau_i \in I\}} \big([\![w, j \models \psi]\!] \wedge \textstyle\bigwedge_{i \leq k < j} [\![w, k \models \varphi]\!]\big)
\end{aligned}$$

Furthermore, for $\tau \in \mathbb{Q}_+$, let $[\![w \models \varphi]\!]^\tau := [\![w, \mathrm{tp}(w, \tau) \models \varphi]\!]$ if $\mathrm{tp}(w, \tau)$ is defined, and $[\![w \models \varphi]\!]^\tau := \bot$, otherwise. Note that we abuse notation here and unify MTL's constant t with the Boolean value $\mathsf{t} \in \mathsf{3}$, and MTL's connectives $\neg$ and $\vee$ with the corresponding three-valued operators in Table 1. Also note that when propositions are only assigned to Boolean values, i.e., $w \in TW^\omega(\Gamma)$ with $\Gamma := 2^P$ then the above definition coincides with MTL's standard two-valued semantics.

We use standard syntactic sugar, e.g., $\varphi \to \psi$ abbreviates $(\neg\varphi) \vee \psi$, and $\Diamond_I \varphi$ ("eventually") and $\Box_I \varphi$ ("always") abbreviate $\mathsf{t}\,\mathsf{U}_I\, \varphi$ and $\neg\Diamond_I \neg\varphi$, respectively. The past-time counterparts $\blacklozenge_I \varphi$ ("once") and $\blacksquare_I \varphi$ ("historically") are defined as expected. The nonmetric variants of the temporal connectives are also easily defined, e.g., $\Box\varphi := \Box_{[0,\infty)}\varphi$. Finally, we use standard conventions concerning the connectives' binding strength to omit parentheses.

▶ **Example 1.** The formula $\Box\, \mathit{req} \to \Diamond_{[0,100)}\, \mathit{ack}$ expresses a simple deadline property of a request-response protocol between two system components. Namely, requests must be acknowledged within 100 milliseconds, assuming that the unit of time is milliseconds.

3 Monitoring Architecture

The target system we monitor consists of one or more system components. The objective of monitoring is to determine at runtime whether the system's behavior, as observed and reported by the system components, satisfies a given MTL specification φ at some or all time points. To this end, we extend the system with additional components called *monitors*. The system components communicate with the monitors and the monitors communicate with each other. Communication takes place over channels. In the following, we explain the system assumptions, sketch the design of our monitoring extension, and state the monitors' requirements.

System Assumptions. We make the following assumptions on our system model.

A 1. *The system is static.*
This means that no system components are created or removed at runtime. Furthermore, each monitor is aware of the existence of all the system components. Note that this assumption can easily be eliminated by building into our algorithm a mechanism to register components before they become active and unsubscribing them when they become inactive. To register components we can, e.g., use a simple protocol where a component sends a registration request and waits until it receives a message that confirms the registration.

A 2. *Communication between components is asynchronous and unreliable. However, messages are neither tampered with nor delivered to wrong components.*
Asynchronous, unreliable communication means that messages may be received in an order different from which they were sent, and some messages may be lost and therefore never received. Note that message loss covers the case where a system component

crashes without recovery. A component that stops executing is indistinguishable to other processes from one that stops sending messages or none of its messages are received. We explain in Remark 7 in Section 4 that it is also straightforward to handle the case where crashed processes can recover. The assumption ruling out tampering and improper delivery can be discharged in practice by adding information to each message, such as a recipient identifier and a cryptographic hash value, which are checked when receiving the message.

A 3. *System components, including the monitors, are trustworthy.*
This means, in particular, that the components correctly report their observations and do not send bogus messages.

A 4. *Observations about a proposition's truth value are consistent.*
This means that no components observe that a proposition $p \in P$ is both true and false at a time $\tau \in \mathbb{Q}_+$.

A 5. *The system components make infinitely many observations in the limit.*
This guarantees that the observable system behavior is an infinite timed word. Note that MTL formulas specify properties about infinite timed words. We would need to use another language if we want to express properties about finite system behavior. However, note that a monitor is always aware of only a finite part of the observed system behavior. Furthermore, since channels are unreliable and messages can be lost, a monitor might even, in the limit, be aware only of a finite part of the infinite system behavior.

System Design. The monitors are organized in a directed acyclic graph structure, where each monitor is responsible for some subformula of the given MTL specification φ. The decomposition of φ into the subformulas used for monitoring is system and application specific. However, we require that it respects the subformula ordering in that if the monitor M' is in the subgraph of the monitor M, then the formula that M' monitors is a subformula of the one monitored by M. Moreover, the monitor at the root is responsible for φ. It outputs verdicts of the form $(b, \tau) \in \mathsf{2} \times \mathbb{Q}_+$, with the meaning that φ has the truth value b at time τ. We also add a unidirectional communication channel from each monitor to its parent monitors and unidirectional communication channels from the system components to the monitors. The system components are instrumented to send their observations to the monitors. This instrumentation is also system and application specific, and irrelevant for the functioning of the monitors; hence we do not discuss it further.

Three types of messages are exchanged during monitoring: report, notify, and alive.

- A system component sends the message $\mathsf{report}(p, b, \tau)$ when it observes at time $\tau \in \mathbb{Q}_+$ that the Boolean value $b \in \mathsf{2}$ is assigned to the proposition $p \in P$. This message is only sent to the monitors that are responsible for a subformula ψ of φ in which p occurs in one of ψ's subformulas for which no other monitor is responsible. Analogously, a monitor responsible for ψ sends messages of the form $\mathsf{report}(\psi, b, \tau)$ to inform its parent monitors about verdicts (b, τ) for the subformula ψ of φ.
- A system component C sends the message $\mathsf{notify}(C, \tau, s)$ to all monitors to inform them about some observation at time $\tau \in \mathbb{Q}_+$. The need to send such messages originates from MTL's *point-based* semantics. Their purpose is that all monitors are aware of all the time points and their timestamps of the timed word representing the observed system behavior. This message includes a sequence number $s \in \mathbb{N}$, which is the number of notify messages that C has sent so far, including the current one. A monitor uses s to determine whether it knows all time points up to time τ.

- A system component C can also send the message $\mathsf{alive}(C, \tau, s)$ when it has not made any observations for a while. The sequence number s is the number of messages of the form $\mathsf{notify}(C, \tau', s')$ with $\tau' < \tau$ that have been sent by C. The alive messages help a monitor to determine whether it has received all notify messages over some time period. In particular, alive messages are handy when components have not made any observations for a while.

▶ **Remark 2.** In what follows, we assume that there is only a single monitor. By this assumption, there are no messages of the form $\mathsf{report}(\psi, b, \tau)$, which are sent by a monitor responsible for the subformula ψ of φ. This assumption is without loss of generality since we can replace ψ in φ by a fresh proposition p_ψ and consider the submonitor as yet another system component. Note that this component need not send notify messages about the existence of time points since they are already sent by the other system components.

Monitor Requirements. Let O be the set of messages corresponding to the observations made by the system components and therefore, by A3, sent to (but not necessarily received by) the monitors. We use the timed word $w(O)$ to model the observable system behavior. It satisfies the following conditions.

(i) For every $\mathsf{notify}(C, \tau, s) \in O$, there is a letter (σ, τ) in $w(O)$.
(ii) For every $\mathsf{report}(p, b, \tau) \in O$, there is a letter (σ, τ) in $w(O)$ with $\sigma(p) = b$.
(iii) For every letter (σ, τ) in $w(O)$, there is some $\mathsf{notify}(C, \tau, s) \in O$ and for all $p \in P$, if $\sigma(p) \neq \bot$ then $\mathsf{report}(p, \sigma(p), \tau) \in O$.

Note that $w(O)$ is uniquely determined by the notify and report messages in O. First, for each $\tau \in \mathbb{Q}_+$, there is at most one letter with the timestamp τ in a timed word. Hence, all notify and report messages that include the timestamp τ determine the letter in $w(O)$ with this timestamp. The letter's position in $w(O)$ is also determined by τ. Second, because of A4, $\mathsf{report}(p, b, \tau) \in O$ implies $\mathsf{report}(p, \neg b, \tau) \notin O$. Finally, by A5, $w(O)$ is infinite.

We state the requirements of our monitoring approach concerning its correctness with respect to $w(O)$. The messages are processed iteratively by a monitor M for the formula φ and it keeps state between iterations. M's input in an iteration is a message and its output is a set $V \subseteq 2 \times \mathbb{Q}_+$ of verdicts. We denote M's output after processing a message m by $M(m)$. Let $\bar{m} = m_0, m_1, \ldots$ be a sequence of messages from O of length $N \in \mathbb{N} \cup \{\infty\}$.

- A monitor M is *sound* for φ on $\bar{m}$ if for all $\tau \in \mathbb{Q}_+$ and $b \in 2$, if $(b, \tau) \in M(m_i)$ for some $i < N$, then $[\![w(O) \models \varphi]\!]^\tau = b$.
- A monitor M is *complete* for φ on $\bar{m}$ if for all $\tau \in \mathbb{Q}_+$, and $b \in 2$, if $[\![w(O) \models \varphi]\!]^\tau = b$ then $(b, \tau) \in M(m_i)$, for some $i < N$.

▶ **Remark 3.** Completeness together with soundness is not achievable in general. One reason is failures, cf. A2. For instance, if all messages are lost, it is only possible in trivial cases for a monitor to soundly output verdicts for every violation. We therefore require completeness of a monitor only under the assumption that every message in O is eventually received by the monitor and the monitor never crashes. Another reason is that not all formulas are "monitorable" [17]. For example, the formula $\Box\Diamond p$, which states that p is true infinitely often, can only be checked on $w(O)$. However, a monitor only knows finite parts of $w(O)$ at any time, which is insufficient to determine whether the formula is fulfilled or violated. To simplify matters, we focus in the forthcoming sections on *bounded* formulas, i.e., the metric constraint of any temporal future-time connective is a finite interval. Note that if the formula ψ is bounded then $\Box\psi$ describes a safety property. Many deadline requirements have this form. Since we consider verdicts for all $\tau \in \mathbb{Q}_+$ with $[\![w(O) \models \psi]\!]^\tau \in 2$, the outermost temporal connective $\Box$ is implicitly handled by a monitor.

▶ **Example 4.** Consider a system with a single component C. Let O be an infinite set of messages containing the messages $\mathsf{notify}(C, 0.5, 1)$, $\mathsf{report}(p, \mathsf{f}, 0.5)$, $\mathsf{notify}(C, 2.0, 2)$, and $\mathsf{report}(p, \mathsf{f}, 2.0)$, and no other message with a timestamp less than or equal to 2.0. Note that the sequence number of the first notify message that C sends is 1 since C's sequence-number counter is incremented before C sends the message. Furthermore, assume that the message $\mathsf{report}(p, \mathsf{f}, 0.5)$ is lost, while all other messages are received by the monitor. A sound monitor for the formula $\blacklozenge_{[0,1]}\, p$ can at most output the verdicts $(\mathsf{f}, 0.5)$ and $(\mathsf{f}, 2.0)$ for the time points 0 and 1, respectively. However, since a monitor does not know p's truth value at time 0.5, it cannot deduce the verdict $(\mathsf{f}, 0.5)$, and is therefore incomplete. Note that a monitor can deduce the verdict $(\mathsf{f}, 2.0)$ because, from the sequence numbers of the notify messages, it can infer that there is no other time point originating from the component C in $w(O)$ with a timestamp between 1.0 and 2.0.

4 Verdict Computation

In this section, we explain how a monitor processes a sequence of messages from the set O of messages sent by the system components and how it computes verdicts.

Main Loop. The monitor's main procedure Monitor, given in Figure 1, is invoked for each message received. It takes as input φ, the formula to be monitored, and a message. It updates the monitor's state, thereby computing verdicts, which it returns. The verdicts computed in an iteration of the monitor are stored in the global variable verdicts, which is set to the empty set at the start of processing the received message.

```
procedure Monitor(φ, msg)
  verdicts ← ∅
  case msg = notify(_, τ, _)
    NewTimePoint(φ, τ)
  case msg = report(p, b, τ)
    NewTimePoint(φ, τ)
    SetTruthValue((p, {τ}), b)
  foreach J in NewCompleteIntervals(msg) do
    NoTimePoint(φ, J)
  return verdicts
```

Figure 1 The monitor's main loop.

Intuitively speaking, with each received message the monitor gains knowledge about the infinite timed word $w(O)$. The monitor's partial knowledge about $w(O)$ is reflected in the monitor's state. The monitor's state is maintained by the procedures NewTimePoint, SetTruthValue, and NoTimePoint. When a $\mathsf{notify}(C, \tau, s)$ message is received, Monitor calls the NewTimePoint procedure, which makes the monitor aware of the existence of the time point with the timestamp τ in $w(O)$. When a $\mathsf{report}(p, \tau, b)$ message is received, Monitor calls the SetTruthValue procedure, which sets the proposition p's truth value at the time point with timestamp τ to the Boolean value b. It also deduces, whenever possible, the truth values of φ's subformulas at the known time points in $w(O)$. This deduction can result in new verdicts. Note that, prior to SetTruthValue, Monitor calls the NewTimePoint procedure, which ensures that the monitor is aware of the existence of the time point with timestamp τ in $w(O)$. Finally, Monitor accounts for the intervals that became complete by the received message. We say that an interval $J \subseteq \mathbb{Q}_+$ is *complete* if the monitor has received all notify messages with a timestamp in J from all system components. In particular, if J is incomplete, then the monitor does not yet know all the timestamps in J from letters in $w(O)$. The procedure NewCompleteIntervals returns new complete intervals, based on the sequence number of the received message and the monitor's state. Note that only notify and alive messages contain a sequence number; for a report message, NewCompleteIntervals does not return any intervals. For each of the returned intervals, the NoTimePoint procedure updates the monitor's state accordingly.

In the following, we provide some details about the monitor's state and how it is updated. We start by explaining the main data structure stored in the monitor's state.

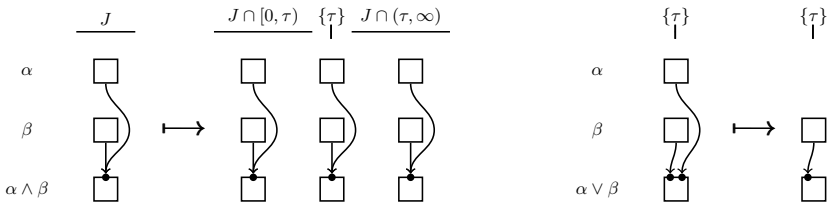
Data Structure. The main data structure is a graph structure. Its *nodes* are pairs of the form (ψ, J), where ψ is a subformula of the monitored formula φ and $J \subseteq \mathbb{Q}_+$ is an interval. The interval J is either a singleton $\{\tau\}$, where τ is the timestamp occurring in a received notify or report message (thus τ occurs in a letter in $w(O)$), or it is an incomplete interval. Initially, there is a node $(\psi, [0, \infty))$ for each subformula ψ of φ. The interval $[0, \infty)$ corresponds to the fact that no time points have been created yet. Each node is associated with a truth value, initially $\bot$. Furthermore, each node contains a set of *guards* and a set of outgoing pointers to guards, called *triggers*. We call the source node of an incoming pointer to a guard a *precondition*. Intuitively, a guard with no preconditions (i.e., no incoming pointers) is satisfied and we assign the Boolean value t to the guard's node; if a node has no guards, we assign the Boolean value f to the node; otherwise, if a node has guards with incoming pointers, the node is assigned the truth value $\bot$. Overall, the graph structure can be viewed as an AND-OR-graph, where intuitively a node's truth value is given by the disjunction over the node's guards of conjunctions of the truth values of each guard's preconditions.

Updates. The first time the monitor receives a notify or a report message with some timestamp τ, a new time point in $w(O)$ is identified and the data structure is updated. Note that the timestamp τ is necessarily in some incomplete interval J. Each node (ψ, J) in the graph is replaced by the nodes $(\psi, \{\tau\})$, $(\psi, J \cap [0, \tau))$, and $(\psi, J \cap (\tau, \infty))$. The links of the new nodes to and from the other nodes are created based on the links of the node (ψ, J). Links are used to propagate Boolean values from one node to another when, for example, receiving a report message. These two tasks, creating nodes and propagating truth values, are carried out by the procedures NewTimePoint and SetTruthValue, respectively. NewTimePoint also deletes a node (ψ, J) after creating the new nodes for the split interval J, and SetTruthValue deletes nodes when they are no longer needed for propagating truth values. A call to SetTruthValue$((\varphi, \{\tau\}), b)$, for some timestamp τ and Boolean value b, also adds the verdict (b, τ) to the set verdicts.

When the monitor infers that a nonsingular interval J is complete, it calls the procedure NoTimePoint, which deletes the nodes of the form (ψ, J) and updates triggers if necessary. Moreover, it calls the procedure SetTruthValue when a Boolean value can be assigned to a node. The monitor uses the sequence numbers in notify and alive messages to determine whether there are no time points with timestamps in J. For such a J, the monitor must have received from each component C messages of one of the forms: (1) notify(C, τ, s) with $\tau \leq \inf J$ and either notify$(C, \tau', s+1)$ or alive(C, τ', s) with $\tau' \geq \sup J$, or (2) alive(C, τ, s) with $\tau \leq \inf J$ and either notify$(C, \tau', s+1)$ or alive(C, τ', s) with $\tau' \geq \sup J$. In the latter case, we assume without loss of generality that the monitor has received at the beginning the message alive$(C, -1.0, 0)$ from each component C.

In the following, we explain how nodes are created and Boolean values are propagated. For a newly created node (ψ, J), its guards and their preconditions depend on ψ's main connective. We first focus on the simpler cases where the main connective is nontemporal.

A node for the formula $\psi = \alpha \vee \beta$ has two guards, each with a precondition for ψ's direct subformulas. Analogously, when considering $\wedge$ as a primitive, the node for $\psi = \alpha \wedge \beta$ has one guard with two preconditions for the two direct subformulas. A node for the formula $\psi = \neg\alpha$ has one guard with the node for the formula α as the precondition associated to the same time point or an incomplete interval. The first two cases are illustrated on the



(a) node creation for a new time point (b) propagation

Figure 2 Adjusting guards in case of (a) the creation of a new time point at $\tau \in J$ for the formula $\alpha \wedge \beta$, and (b) propagation, namely when the node $(\alpha, \{\tau\})$ is set to f, for the formula $\alpha \vee \beta$.

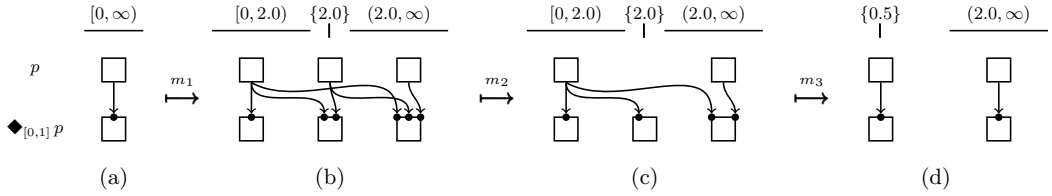


Figure 3 The data structure (a) before receiving any message, and after receiving the messages (b) $m_1 = \mathsf{notify}(C, 2.0, 2)$, (c) $m_2 = \mathsf{report}(p, \mathsf{f}, 2.0)$, and (d) $m_3 = \mathsf{notify}(C, 0.5, 1)$.

left-hand side of the arrow $\mapsto$ of Figure 2(a) and (b), respectively. A box corresponds to a node, where the node's formula is given by the row and the interval by the column of the box. Dots correspond to guards and arrows to triggers. Figure 2(a) also illustrates how the data structure is updated when a new time point is added; in the case of Boolean connectives, this is done by simply duplicating the nodes and their guards and triggers. The creation of the guards of a node for a formula with the main connective S_I or U_I is more complex as the preconditions are nodes that can be associated to time points or incomplete intervals different from the node's interval J. We first sketch how Boolean values are propagated before explaining these more complex cases.

When receiving a $\mathsf{report}(p, b, \tau)$ message, we set the truth value of the node $(p, \{\tau\})$ to the Boolean value b, provided that the node exists. This value is then propagated through the node's triggers to its successor nodes. However, for negation, the Boolean value propagated from a node (α, J) to the node $(\neg\alpha, J)$ is the complement of the Boolean value associated to the node (α, J). The propagation of the Boolean value t corresponds to deleting just the triggers, whereas the propagation of f corresponds to also deleting the guards that the triggers point to. If a guard of a successor node has no more preconditions, then we set the successor's nodes value to t; in contrast, if the set of guards of a successor node becomes empty, then we set its value to f. Figure 2(b) illustrates the propagation of a truth value through the data structure for the simple case where the formula is of the form $\alpha \vee \beta$.

Temporal Connectives. Before describing the general case of handling formulas ψ of the forms $\alpha\, \mathsf{S}_I\, \beta$ and $\alpha\, \mathsf{U}_I\, \beta$, we consider a simpler example where $\psi = \blacklozenge_I\, p$. In this case, a node $(\blacklozenge_I\, \psi, J)$ has a guard for every node (ψ, K) for which there are a $\tau \in J$ and $\kappa \in K$ such that $\tau - \kappa \in I$. Each guard has exactly one precondition, namely the corresponding node (ψ, K).

▶ **Example 5.** We reconsider the formula $\varphi = \blacklozenge_{[0,1]}\, p$ from Example 4, and a set O that contains the messages $m_1 := \mathsf{notify}(C, 2.0, 2)$, $m_2 := \mathsf{report}(p, \mathsf{f}, 2.0)$, and $m_3 := \mathsf{notify}(C, 0.5, 1)$. We assume that the monitor receives m_1, m_2, and m_3 in this order. Figure 3 illustrates how

the data structure is updated after receiving each of these messages. The updates performed after the first two messages are clear from the previous explanations, whereas the update performed for the message m_3 comprises the following update steps. First, the interval $[0, 2.0)$ is split at timestamp 0.5, thus deleting the nodes $(p, [0, 2.0))$ and $(\varphi, [0, 2.0))$, and creating six new nodes, together with their guards and triggers. Namely, a node is created for each of the formulas p and φ, and for each of the intervals $[0, 0.5)$, $\{0.5\}$, and $(0.5, 2.0)$. The preconditions of the other nodes are also updated accordingly to the intervals of the newly created nodes. Second, the four nodes corresponding to the intervals $[0, 0.5)$ and $(0.5, 2.0)$, together with their triggers, are deleted. This is because these intervals are complete, that is, no time point of $w(O)$ has a timestamp in these intervals. By deleting these four nodes, the node $(\varphi, \{2.0\})$ remains with no guards. Indeed, after the split, the node remains with only one guard, which has the precondition $(p, (0.5, 2.0))$. The other nodes (p, J) are not preconditions because no timestamp in those intervals J satisfies the temporal constraint. This single guard is deleted when its precondition $(p, (0.5, 2.0))$ is deleted. Finally, as the node $(\varphi, \{2.0\})$ is without guards, it is assigned the Boolean value f. These steps lead to the structure in Figure 3(d).

We now consider the general case where the formula ψ is $\alpha \,\mathsf{S}_I\, \beta$; the case for $\alpha \,\mathsf{U}_I\, \beta$ is dual. The node (ψ, J) has a guard for each node (β, K) with the truth value t or $\bot$, and with $(J \ominus K) \cap I \neq \emptyset$, where $J \ominus K := \{\tau - \kappa \mid \tau \in J \text{ and } \kappa \in K\}$. Moreover, for each such node (β, K) and all nodes (α, H), with H between K and J, we have that (α, H) is not assigned to the truth value f. We call the node (β, K) an *anchor node* for the node (ψ, J), and a node (α, H) a *continuation node* for the anchor (β, K). The node (α, H) must be strictly after (β, K) if K is a singleton, but we can have $H = K$ otherwise. Note too that a node can be a continuation node for multiple anchor nodes. A guard has a trigger from its anchor node if the truth value assigned to the anchor node is $\bot$. Furthermore, the guard has a trigger from the first continuation node after the anchor node that is assigned to the truth value $\bot$. If this continuation node is assigned to the Boolean value t at a later time, we move the trigger to the second such continuation node, and delete it if such a node does not exist. Alternatively, we could unroll ψ into a disjunction of conjunctions and use the guard constructions presented previously for $\wedge$ and $\vee$. However, each guard would have multiple continuation nodes as preconditions, which would result in an unnecessary overhead.

When splitting the interval J at time τ, we create the new nodes (ψ, J'), with $J' \in \{J \cap [0, \tau), \{\tau\}, J \cap (\tau, \infty)\}$, together with the nodes' guards. For this, we use the guards of the node (ψ, J). After creating the new nodes, we delete the node (ψ, J). The split preserves the invariant, stated in the previous paragraph, about the nodes' guards. This invariant is key in the algorithm's soundness proof. We illustrate the construction for the specific case depicted on the left-hand side of Figure 4. There are two guards of $(\alpha \,\mathsf{S}_I\, \beta, J)$, each with two preconditions. The first guard has the anchor node (β, K) and the continuation node (α, H). The second guard has the anchor node (β, J) and the continuation node (α, J). The triggers of the first guard are drawn with solid lines in Figure 4 and the triggers of the second guard are drawn with dashed lines. We assume that J, K, and H are pairwise disjoint. We also assume that $0 \in I$ and that the metric constraint is satisfied for the new nodes and their anchors. The right-hand side of Figure 4 shows the guards for the new nodes along with their triggers.

Initialization. The monitor's state is initialized by the procedure Initialize, which takes φ as argument. We assume that it is called before processing the received messages by the Monitor procedure. Initially, the nodes of the graph structure are $(\psi, [0, \infty))$, where ψ is a subformula

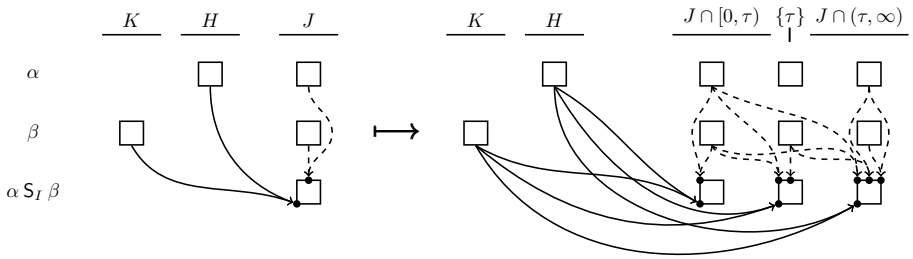


Figure 4 Adjusting guards when creating a new time point at $\tau \in J$ for the formula $\alpha \, \mathsf{S}_I \, \beta$.

of φ, with the corresponding guards and triggers. The truth value of a node $(\psi, [0, \infty))$ is $\bot$, except for the node $(\mathsf{t}, [0, \infty))$, which has the truth value t. Note that the node $(\mathsf{t}, [0, \infty))$ only exists if the constant t occurs in φ. For this node, we invoke the procedure SetTruthValue to propagate its Boolean value.

Correctness Guarantees. The correctness guarantees of the monitoring algorithm are given in the following theorem.

▶ **Theorem 6.** *Let $\bar{m} = m_0, m_1, \ldots$ be the sequence of messages in O received by the monitor.*

(i) *The monitor is sound for φ on $\bar{m}$.*

(ii) *The monitor is complete for φ on $\bar{m}$, if (a) all temporal future connectives in φ are bounded (i.e., their metric constraints are finite intervals), and (b) for every $m \in O$, there is some $i \in \mathbb{N}$ with $m_i = m$.*

▶ **Remark 7.** When a process crashes, its state is lost. To recover a process we must bring it into a state that is safe for the system. To safely restart a system component, we must restore its sequence number. We can use any persistent storage available to store this number. In case the component crashes while storing this number, we can increment the restored number by one. This might results in knowledge gaps for some monitors, since some intervals will never be identified as complete. However, the computed verdicts are still sound.

For the recovery of a crashed monitor, we just need to initialize it. In particular, the nodes of its graph structure are of the form $(\psi, [0, \infty))$, where ψ is a subformula of φ. A recovered monitor corresponds to a monitor that has not yet received any messages. This is safe in the sense that the recovered monitor will only output sound verdicts. When the monitor also logs received messages in a persistent storage, it can replay them to close some of its knowledge gaps. Note that the order in which these messages are replayed is irrelevant and they can even be replayed whenever the recovered monitor is idle.

5 Accuracy of Timestamps

The monitors' verdicts are computed with respect to the observations that the monitors receive from the system components. These observations might not match with the actual system behavior. In particular, the timestamp in a message $\mathsf{report}(p, b, \tau)$ may be inaccurate because τ comes from the clock of a system component that has drifted from the actual time. Nevertheless, we use these timestamps to determine the time between observations. Hence, one may wonder in what sense are the verdicts meaningful.

Consider first the guarantees we have under the additional system assumption that timestamps are precise and from the domain $\mathbb{Q}_+$. Under this assumption, $w(O) \preceq w$, where $w(O) \in TW^\omega(\Sigma)$ is the observed system behavior and $w \in TW^\omega(\Gamma)$ represents the real system behavior. Note that in w, all propositions at all time points are assigned Boolean values, which might not be the case in $w(O)$ since no system component observes whether a proposition is true or false at a time point. It follows from Lemma 8 that the verdicts computed from the observed system behavior $w(O)$ are also valid for the system behavior w.

▶ **Lemma 8.** *Let φ be an MTL formula, $v, v' \in TW^\omega(\Sigma)$, and $\tau \in \mathbb{Q}_+$. If $v \preceq v'$ then $[\![v \models \varphi]\!]^\tau \preceq [\![v' \models \varphi]\!]^\tau$.*

Assuming precise timestamps is however a strong assumption, which does not hold in practice since real clocks are imprecise. Moreover, each system component uses its local clock to timestamp observations and these clocks might differ due to clock drifts. In fact, assuming synchronized clocks boils down to having a synchronized system at hand.

Nevertheless, we argue that for many kinds of policies and systems, relying on timestamps from existing clocks in monitoring is good enough in practice. First, under stable conditions (like temperature), state-of-the-art hardware clocks already achieve a high accuracy and their drifts are, even over a longer time period, rather small [7]. Moreover, there are protocols like the Network Time Protocol (NTP) [16] for synchronizing clocks in distributed systems that work well in practice. For local area networks, NTP can maintain synchronization of clocks within one millisecond [14]. Overall, with state-of-the-art techniques, we can obtain timestamps that are "accurate enough" for many monitoring applications, for instance, for checking whether deadlines are met when the deadlines are in the order of seconds or even milliseconds. Furthermore, if the monitored system guarantees an upper bound on the imprecision of timestamps, we can often account for this imprecision in the policy formalization. For example, if the policy stipulates that requests must be acknowledged within 100 milliseconds and the imprecision between two clocks is always less than a millisecond, then we can use the MTL formula $\Box\, \mathit{req} \rightarrow \blacklozenge_{[0,1)} \Diamond_{[0,101)}\, \mathit{ack}$ to avoid false alarms.

6 Related Work

Multi-valued semantics for temporal logics are widely used in monitoring, see e.g., [5, 4, 3, 18, 15]. Their semantics extend the classical LTL semantics by also assigning non-Boolean truth values to finite prefixes of infinite words. The additional truth values differentiate whether some or all extensions of a finite word satisfy a formula. However, in contrast to the three-valued semantics of MTL used in this paper, the Boolean and temporal connectives are not extended over the additional truth values. Furthermore, the partial order $\prec$ on the truth values, which orders them in knowledge, is not considered. Note that having the third truth value $\bot$ at the logic's object level and the partial order $\prec$ is at the core of our monitoring approach, namely it is used account for a monitor's knowledge gaps. Multi-valued semantics for temporal logics have also been considered in other areas of system verification. For example, Chechik et al. [6] describe a model-checking approach for a multi-valued extension for the branching-time temporal logic CTL. Their CTL extension is similar to our extension of MTL in the sense that it allows one to reason about uncertainty at the logic's object level. However, the considered tasks are different. Namely, in model checking, the system model is given—usually finite-state—and correctness is checked offline with respect to the model's described executions; in contrast, in runtime verification, one checks online the correctness of the observed system behavior.

Several monitoring algorithms have been developed for verifying distributed systems at runtime [19, 18, 3, 8, 15]. They make different assumptions on the system model and thus target different kinds of distributed systems. Furthermore, they handle different specification languages. None of them account for network failures or handle specifications with real-time constraints. Sen et al. [19] use an LTL variant with epistemic operators to express distributed knowledge. The verdicts output by the monitors are correct with respect to the local knowledge the monitors obtained about the systems' behavior. Since their LTL variant only comprises temporal connectives that refer to the past, only safety properties are expressible. Scheffel and Schmitz [18] extend this work to also handle some liveness properties by working with a richer fragment of LTL that includes temporal connectives that also refer to the future. The algorithm by Bauer and Falcone [3] assumes a lock-step semantics and thus only applies to synchronous systems. Falcone et al. [8] weaken this assumption. However, each component must still output its observations at each time point, which is determined by a global clock. The observations are then received by the monitors at possibly later time points. The algorithm by Mostafa and Bonakdarbour [15] assumes lossless FIFO channels for asynchronous communication. Logical clocks are used to partially order messages.

Various monitoring algorithms have been developed, analyzed, and used to verify real-time constraints at runtime, see e.g., [20, 5, 13, 2]. All of them, however, fall short for monitoring distributed systems. For instance, they do not account for out-of-order message deliveries and the monitor's resulting knowledge gaps about the observed system behavior. It is this shortcoming of prior work that motivated us to develop the monitoring algorithm presented in this paper.

7 Conclusion

We have presented a monitoring algorithm for verifying the behavior of a distributed system at runtime, where properties are specified in the real-time logic MTL. Our algorithm accounts for failures and out-of-order message deliveries. The monitors' verdicts are sound with respect to the observed system behavior. In particular, timestamps originating from local clocks determine the time between the observations made by the system components and sent to the monitors. Note that the ground truth for system behavior is not accessible to the monitors because the monitors themselves are system components.

There are several directions for extending our work. First, we have considered a monitor's completeness from a global perspective, i.e., the observable system behavior. An alternative would be with respect to the knowledge a monitor can infer from the messages it receives. We intend to investigate when a monitor is complete under this perspective. Second, we have opted for a point-based semantics for MTL. An alternative is to use an interval-based semantics, which can be more natural but it also makes monitoring more complex, see [2]. Future work is to adapt the presented monitoring algorithm to an interval-based semantics. Finally, we plan to evaluate our monitoring algorithm on a substantial case study.

Acknowledgments. We thank Srdjan Marinovic for his input and for many helpful discussions in the early phase of this work.

References

1 R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the 1991 REX Workshop on Real Time: Theory in Practice*, volume 600 of *Lect. Notes Comput. Sci.*, pages 74–106. Springer, 1992.

2 D. Basin, F. Klaedtke, and E. Zălinescu. Algorithms for monitoring real-time properties. In *Proceedings of the 2nd International Conference on Runtime Verification (RV)*, volume 7186 of *Lect. Notes Comput. Sci.*, pages 260–275. Springer, 2011.

3 A. Bauer and Y. Falcone. Decentralised LTL monitoring. In *Proceedings of the 18th International Symposium on Formal Methods (FM)*, volume 7436 of *Lect. Notes Comput. Sci.*, pages 85–100. Springer, 2012.

4 A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL semantics for runtime verification. *J. Logic Comput.*, 20(3):651–674, 2010.

5 A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Meth.*, 20(4):14, 2011.

6 M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. Multi-valued symbolic model-checking. *ACM Trans. Softw. Eng. Meth.*, 12(4), 2003.

7 F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):642–657, 1999.

8 Y. Falcone, T. Cornebize, and J.-C. Fernandez. Efficient and generalized decentralized monitoring of regular languages. In *Proceedings of the 34th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, volume 8461 of *Lect. Notes Comput. Sci.*, pages 66–83. Springer, 2014.

9 M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

10 S. C. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, 1950.

11 R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, 1990.

12 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

13 O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS) and on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 3253 of *Lect. Notes Comput. Sci.*, pages 152–166. Springer, 2004.

14 D. L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Netw.*, 3(3):245–254, 1995.

15 M. Mostafa and B. Bonakdarbour. Decentralized runtime verification of LTL specifications in distributed systems. In *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 494–503. IEEE Computer Society, 2015.

16 Network time protocol. `www.ntp.org`, webpage accessed on March 26, 2015.

17 A. Pnueli and A. Zaks. PSL model checking and run-time verification via testers. In *Proceedings of the 14th International Symposium on Formal Methods (FM)*, volume 4085 of *Lect. Notes Comput. Sci.*, pages 573–586. Springer, 2008.

18 T. Scheffel and M. Schmitz. Three-valued asynchronous distributed runtime verification. In *Proceedings of the 12th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMCODE)*, pages 52–61. IEEE Computer Society, 2014.

19 K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 418–427. IEEE Computer Society, 2004.

20 P. Thati and G. Roşu. Monitoring algorithms for metric temporal logic specifications. In *Proceedings of the 4th Workshop on Runtime Verification (RV)*, volume 113 of *Elec. Notes Theo. Comput. Sci.*, pages 145–162. Elsevier Science Inc., 2005.