37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2017, December 12–14, 2017, IIT Kanpur, India

Edited by

Satya Lokam

R. Ramanujam

Editors

Satya Lokam	R. Ramanujam
Microsoft Research	Institute of Mathematical Sciences
Bangalore	Chennai
satya@microsoft.com	jam@imsc.res.in

ACM Classification 1998
F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity, F.3 Logics and Meanings of Programs, F.4 Mathematical Logic and Formal Languages, G.1.3 Numerical Linear Algebra, G.1.6 Optimization

ISBN 978-3-95977-055-2

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/978-3-95977-055-2.

Publication date
January 2018

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.2017.0

ISBN 978-3-95977-055-2 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

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

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume constitutes the proceedings of the 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017), held at the Indian Institute of Technology, Kanpur, from December 12 to December 14, 2017. The FSTTCS conferences are organized annually by the Indian Association for Research in Computing Science (IARCS). The proceedings of FSTTCS 2017 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 programme of the conference consisted of 6 invited talks and 41 contributed papers. This volume contains the contributed papers and extended abstracts of invited talks presented at the conference. The 41 contributed papers were selected from a total of 143 submissions. We thank the programme committee for its efforts in carefully evaluating the submissions and selecting these papers to put together a good programme. We also thank the external reviewers for sending their informative and timely reviews. Further, we thank all those who submitted their papers to FSTTCS 2017.

We are especially thankful to the invited speakers: Edith Elkind (University of Oxford, UK), Sham Kakade (University of Washington, USA), Anca Muscholl (LaBRI & Université Bordeaux, France), Devavrat Shah (MIT, USA), Vinod Vaikuntanathan (MIT CSAIL, USA) and Thomas Wilke (Christian-Albrechts-Universität zu Kiel, Germany), for accepting our invitations and inspire the FSTTCS community.

The conference was greatly enriched by three satellite events: a pre-conference workshop on *Probabilistic reasoning and Formal Methods* organized by S. Akshay (IIT Bombay) and Kuldip S. Meel (National University of Singapore); a post-conference workshop on *Computational Social Choice Theory* organized by Swaprava Nath (IIT Kanpur) and Sunil Simon (IIT Kanpur); and another post-conference workshop on *Lattice Algorithms and Cryptography* organized by Shweta Agrawal (IIT Madras) and Vinod Vaikuntanathan (MIT, USA). We thank the organizers of these workshops and all the speakers in them.

The organizing committee of the conference from the Department of Computer Science and Engineering, IIT Kanpur, made extensive arrangements for the smooth running of the conference and workshops. We are indebted to them for their invaluable efforts. We thank S.P. Suresh of Chennai Mathematical Institute for work on the conference web page.

We thank Easychair which has become indispensable to our professional lives. Finally, we thank the Dagstuhl LIPIcs staff for their coordination in the production of these proceedings, particularly Marc Herbstritt and Michael Wagner who have been very prompt and helpful in answering our questions.

Satya Lokam and R. Ramanujam
December 2017

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Justified Representation in Multiwinner Voting: Axioms and Algorithms*

Edith Elkind

Department of Computer Science, University of Oxford, Oxford, United Kingdom
elkind@cs.ox.ac.uk

Abstract

Suppose that a group of voters wants to select $k \geq 1$ alternatives from a given set, and each voter indicates which of the alternatives are acceptable to her: the alternatives could be conference submissions, applicants for a scholarship or locations for a fast food chain. In this setting it is natural to require that the winning set represents the voters fairly, in the sense that large groups of voters with similar preferences have at least some of their approved alternatives in the winning set. We describe several ways to formalize this idea, and show how to use it to classify voting rules; surprisingly, two voting rules proposed in the XIXth century turn out to play an important role in our analysis.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity

Keywords and phrases voting, committee selection, axioms, PAV

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.1

Category Invited Paper

1 Setup

The program committee (PC) of a conference in theoretical computer science is about to decide on the set of invited speakers: the chair has solicited nominations from the program committee and the steering committee, and now the PC members have to cast their votes. To keep things simple, the chair sets up a Doodle poll, and asks each PC member to indicate which of the potential speakers they approve of. There are 36 PC members, and four slots for invited talks.

The conference is very broad in scope: it accepts papers on logic, formal methods, algorithms, and complexity. Each PC member identifies with one of these four areas: there are 14 algorithms researchers, 9 experts on formal methods, 8 complexity theorists, and 5 logicians. Each potential speaker is also associated with a particular research area, though some speakers may appeal to researchers in two or more areas. For instance, candidates A, B, C, D and E are supported by almost all algorithms and complexity researchers, in the sense that each of them gets at least 20 votes from PC members who belong to these two communities, candidate F is supported by all formal methods researchers, one algorithms researcher and one complexity theorist, and there are various other candidates who receive 6 or fewer votes; in particular, each logician suggest a different speaker that has no support from other PC members.

How should the chair decide whom to invite given the PC members' votes? Of course, the simplest option is to count how many times each candidate is approved, and select the

* This work was partially supported by European Research Council (ERC) under grant agreement 639945.

four candidates with the maximum number of approvals (breaking ties, say, by seniority or travel costs). However, in our example this means that the chair would end up selecting four candidates who only appeal to "Theory A" (i.e., algorithms and complexity) researchers, and this does not seem fair: if there are four invited talks, and the formal methods community is represented by $9 = 36/4$ PC members, intuitively the formal methods researchers are entitled to at least one slot (which they would probably like to allocate to candidate F). On the other hand, it seems that at least two slots should go to "Theory A" speakers, as they are supported by more than half of the voters. Finding the right balance among these concerns is not easy.

In what follows, we will survey recent research that contributes to our understanding of such scenarios by proposing appropriate axioms and identifying voting rules that satisfy them; we will also discuss the computational complexity of the associated problems. Our presentation is based on a sequence of papers [7, 1, 18, 16, 5, 15, 20] by several overlapping groups of researchers that were published in 2014–2017.

2 Formal Model

We consider a set of n voters $[n] = \{1, \dots, n\}$ who choose among alternatives from the set $A = \{a_1, \dots, a_m\}$. Each voter $i \in [n]$ submits an approval ballot $A_i \subseteq A$, which lists all alternatives that she approves of. The goal is to select a winning set, or *committee*, of size k, where $1 \leq k \leq m$. Throughout this paper, we assume that voters are not strategic.

A *committee selection rule* is a mapping that, given A, k, and the list $(A_i)_{i\in[n]}$, outputs one or more winning committees of size k. Note that we allow ties, but the set of winning committees must be non-empty.

2.1 Examples of Committee Selection Rules

We will now present several committee selection rules, and briefly discuss their properties.

Approval Voting (AV) Under this committee selection rule the score of each alternative $a \in A$ is the number of approvals it gets, i.e., $|\{i \mid a \in A_i\}|$, and the winning committee consists of the k alternatives with the highest scores (if there are ties, we output all committees that can be obtained by breaking these ties in some way). While this is a very simple rule, it may fail to be fair, in the sense that even large minorities of voters may end up not being represented in the winning committee(s). In our example, AV would allocate all four slots to speakers who appeal to the algorithms and complexity community.

Proportional Approval Voting (PAV) Under approval voting, each voter is assumed to assign a 'utility' of one to each of her approved alternatives in the committee. In contrast, under proportional approval voting, we assume that the 'marginal utility' that a voter derives from her j-th approved committee member is $\frac{1}{j}$. Formally, given a committee W and a voter i, we write $u_i(W) = h(|W \cap A_i|)$, where $h(j) = 1 + \frac{1}{2} + \cdots + \frac{1}{j}$, and output all committees W that maximize the total 'utility' $\sum_{i\in[n]} u_i(W)$. In our example the total utility of $\{A, B, C, D\}$ is $20 \times (1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4})$. This is less that the total utility of $\{A, B, C, F\}$, which is $20 \times (1 + \frac{1}{2} + \frac{1}{3}) + 9$, so the PAV rule would allocate a slot to the formal methods speaker. This rule has a very long history: it was proposed by a Danish polymath Thorvald N. Thiele in 1895 [21], and was subsequently rediscovered by other researchers (see, e.g., [10]).

Of course, instead of using harmonic weights $h(\cdot)$, one can use another weight function $w(\cdot)$, i.e., set $u_i(W) = w(|W \cap A_i|)$ for some $w : \mathbb{N} \cup \{0\} \to \mathbb{R}$; we refer to the resulting rule as w-AV. For instance, setting $w(i) = i$ for all $i \in \mathbb{N} \cup \{0\}$ corresponds to Approval Voting. Another interesting option is $w(0) = 0$, $w(i) = 1$ for $i \geq 1$, which is sometimes referred to as Chamberlin–Courant Approval Voting (CCAV). The CCAV rule aims to select committees that represent as many voters as possible, but does not necessarily allocate more representatives to larger groups: in our example, CCAV may select just one of the alternatives in $\{A, B, C, D, E\}$.

Our (and Thiele's) decision to use harmonic weights may seem mysterious at this point, but we will soon see that there are good reasons for it.

Sequential Proportional Approval Voting (SeqPAV) One may wonder if winning committees under PAV can be computed efficiently. It turns out that this is unlikely: deciding if a given committee should win under PAV is coNP-complete [3, 19]. In fact, this was already intuitively obvious to Thiele (more than 70 years before the formal language to talk about such issues was invented), and he proposed a greedy variant of his rule, which is now referred to as Sequential Proportional Approval Voting (SeqPAV). Under this rule, the committee is formed in k steps; we start with an empty committee $W = \emptyset$, and at each step we add a candidate with the largest marginal utility, where the marginal utility of an alternative $a \in A \setminus W$ is computed as

$$\sum_{i \in [n]} \left(u_i(W \cup \{a\}) - u_i(W)\right).$$

In Thiele's original proposal $u_i(W) = h(|W \cap A_i|)$, but the same approach can be used with other weight functions. In particular, under the sequential version of the CCAV rule (SeqCCAV) at each step for each alternative $c \in A \setminus W$ we compute

$$\#c = |\{i \in [n] \mid c \in A_i, A_i \cap W = \emptyset\}|,$$

and add an alternative for which this quantity is maximized.

Monroe Approval Voting (MonAV) and its greedy variant (GrMonAV) Under MonAV, to evaluate a given committee W, we assign scores to all many-to-one matchings between voters and members of W that match each committee member to $\lfloor \frac{n}{k} \rfloor$ or $\lceil \frac{n}{k} \rceil$ voters: the score of a matching is the number of voters who are matched to alternatives they approve. We then select a matching with the maximum score; this score is said to be the score of W. We output committees with the maximum score. Under this rule, $\{A, B, C, F\}$ should be preferred to $\{A, B, C, D\}$: if we choose $\{A, B, C, F\}$, we can match 9 voters to each of A, B, and F, so the score of this committee is at least 27, whereas the score of $\{A, B, C, D\}$ cannot exceed 14+8=22.

Now, we do not need to go over all possible matchings to compute the score of a given committee W: a committee score can be computed in polynomial time using network flow techniques. However, finding a committee with the maximum score is computationally hard [14]. Therefore, it is natural to consider a greedy variant of this rule [9]: again, we proceed in k stages, starting with $W = \emptyset$, and at each stage we select an alternative $c \in A \setminus W$ and $\lfloor \frac{n}{k} \rfloor$ or $\lceil \frac{n}{k} \rceil$ unassigned voters to be matched to this alternative, so that as many of these voters as possible approve c. It is instructive to compare SeqCCAV and GrMonAV: while both rules add an alternative and remove a number of voters associated with this alternative at each step, SeqCCAV only removes voters who approve the selected alternative (and may remove a large number of voters), while GrMonAV removes roughly $\frac{n}{k}$ voters, some of which may not approve the selected alternative.

Rules based on fractional vote transfers One difficulty with the GrMonAV rule is that it matches each voter to a single alternative; if a voter does not approve the alternative she is matched to, naturally we cannot expect her to be happy with the selected committee. To mitigate this difficulty, Sánchez Fernández et al. [15] propose a family of rules, which they call EJR-Exact, that operate similarly to GrMonAV, but may assign a fraction of a voter's weight to a selected alternative. The name of this family of rules may seem mysterious at this point, but it will be explained in the next section. The specific weight assignment rules associated with EJR-Exact rules are fairly complicated, and we will not discuss them here. Another iterative rule that is based on fractional weight assignment was recently proposed by Aziz and Lee [6]; they call their rule the Expanding Approvals Rule (EAR). We note that the idea of fractional assignment of voters to alternatives is quite well-accepted in the context of voting with ranked ballots (where each voters submits an ordering of the alternatives). In fact, the method used to elect the Senate of the Republic of Ireland uses the Single Transferable Vote rule with fractional vote transfers [22].

Phragmén's rule and its sequential variants Under all voting rules described so far, for each voter we define her 'utility' or 'satisfaction' from a given committee, and choose a committee with the aim of maximizing this quantity subject to certain constraints. A Swedish mathematician Lars Edvard Phragmén, who published his work [13] around the same time as Thiele did, proposed a radically different perspective. In his work, each committee member is associated with one unit of *load*, which needs to be distributed among voters who support that alternative as evenly as possible; an alternative's load can only be allocated to voters who approve her, and we aim to minimize the maximum voter load. Thus, for each committee, we find the best load assignment in terms of the maximum voter load and then output committees for which this quantity is minimized. Just as for PAV, finding optimal committees under this rule is NP-hard [8], and Phragmén has developed an iterative version of this rule, which is computationally tractable (again, long before the concept of computational complexity was formalized); the subsequent literature [16] refers to this rule as SeqPhragmen. Another rule that is, in some sense, a dual of Phragmén's rule (or, more precisely, a sequential version of its dual) has been recently proposed by Sánchez Fernández et al. [17], who view their rule as an extension of D'Hondt's method for party list representation, and therefore refer to their rule as Open D'Hondt Method (ODH).

There are many other approval-based committee selection rules; we refer the reader to a survey by Kilgour [11]. However, the rules listed above seem to be particularly well suited to achieve our representation desiderata.

3 Axioms

We will now formulate our first representation axiom; intuitively, this axiom says that there should be a formal methods talk in the program of our conference. In more detail, as there are n voters and k slots, a group of voters of size $\frac{n}{k}$ is 'entitled' to a slot. However, there are many ways of partitioning voters into groups of this size, and it is easy to see that we cannot guarantee representation to every such group. We therefore focus on groups that are *cohesive*, i.e., there exists an alternative approved by all group members; we would like to ensure that no such group is completely disenfranchized. The following axiom was proposed by Aziz and Walsh [7], in a workshop paper that started this line of research.

Justified Representation (JR). A committee W provides *justified representation (JR)* if for every group of voters V such that $|V| \geq \frac{n}{k}$ and $\cap_{i \in V} A_i \neq \emptyset$ there exists a voter $i \in V$ who approves a member of W, i.e., $|A_i \cap W| \geq 1$. A committee selection rule *satisfies justified representation* if every committee in its output provides JR.

It is easy to see that Approval Voting does not satisfy JR; indeed, this is illustrated by our conference example. However, most of the other rules we consider satisfy JR: this is true for PAV (and for every w-AV rule that satisfies $w(1) = 1$, $w(j) - w(j-1) \leq \frac{1}{j}$) [1], for MonAV and GrMonAV [1], as well as for all variants of Phragmén's rule that we have mentioned [8], including the ODH rule [17]. A prominent exception is the SeqPAV rule, which may fail JR [1]. Indeed, sequential variants of almost all weighted AV rules fail JR; the only exception is the sequential variant of the CCAV rule (SeqCCAV) [1].

For most of these rules it is not difficult to prove that they satisfy JR; we provide the argument for two rules to illustrate the main ideas.

▶ **Theorem 1.** *PAV and SeqCCAV satisfy JR.*

Proof. Consider first PAV. Fix a committee W that does not provide JR, i.e., there exists a set of voters V with $|V| \geq \frac{n}{k}$ and $\cap_{i \in V} A_i \neq \emptyset$ such that $A_i \cap W = \emptyset$ for all $i \in V$. Let a be some alternative in $\cap_{i \in V} A_i$. We will argue that there is an alternative $c \in W$ such that $(W \setminus \{c\}) \cup \{a\}$ has higher PAV score than W does, i.e., PAV cannot output W.

Define the *marginal contribution* of an alternative $c \in W$ to be

$$m(c) = \sum_{i \in [n]} \left(u_i(W) - u_i(W \setminus \{c\})\right).$$

By changing the order of summation, we can rewrite the sum of marginal contributions of alternatives in W as

$$\sum_{c \in W} \sum_{i \in [n]} m(c) = \sum_{i \in [n]} \sum_{c \in W} \left(u_i(W) - u_i(W \setminus \{c\})\right). \tag{1}$$

Now, for each voter i with $A_i \cap W = \emptyset$ (so, in particular, for each voter in V) her contribution to the sum in (1) is 0. On the other hand, if $|A_i \cap W| = s > 0$ then for each $c \in A_i \cap W$ we have $u_i(W) - u_i(W \setminus \{c\}) = \frac{1}{s}$ and for each $c \in W \setminus A_i$ we have $u_i(W) - u_i(W \setminus \{c\}) = 0$. Thus, in this case i contributes $s \cdot \frac{1}{s} = 1$ to the sum in (1). As there are at most $n - \frac{n}{k}$ voters who make a non-zero contribution, the sum in (1) can be bounded as $n - \frac{n}{k} < n$, and therefore there is an alternative $c \in W$ with $m(c) < \frac{n}{k}$. Now, consider the committee $(W \setminus \{c\}) \cup \{a\}$. By removing c, we reduce the PAV score by less than $\frac{n}{k}$, and by adding a we increase the PAV score by at least $\frac{n}{k}$, as each voter in V now contributes 1 to the PAV score. This proves our claim.

Now, consider SeqCCAV. Suppose that after k steps it outputs a committee W such that for some set of voters V with $|V| \geq \frac{n}{k}$ and $\cap_{i \in V} A_i \neq \emptyset$ we have $A_i \cap W = \emptyset$ for all $i \in V$. Again, let a be an alternative in $\cap_{i \in V} A_i$. As a was never selected, it was in $A \setminus W$ at each step of the procedure. Moreover, at each iteration we had $\#a \geq \frac{n}{k}$ because of the voters in V. Thus, each of the alternatives added to W was approved by at least $\frac{n}{k}$ previously unrepresented voters, i.e., at each step the number of voters i with $A_i \cap W \neq \emptyset$ grew by at least $\frac{n}{k}$. Thus, after k steps no unrepresented voters should have remained, a contradiction. ◀

By now, the reader may have an impression that essentially any reasonable approval-based committee selection rule satisfies JR. However, this is not quite the case: there are well-established rules, such as, e.g., Satisfaction Approval Voting (SAV) and Minimax Approval

Voting (MAV) that fail this axiom; see, e.g., [11] for the definitions of these rules, and [1] for respective counterexamples. Indeed, the original workshop paper of Aziz and Walsh only identified a single voting rule that satisfied EJR, namely, SeqCCAV. While the authors had considered PAV, they left it as an open problem to show that this rule satisfies JR. This open problem was resolved independently by two groups of researchers (Brill, Conitzer, Freeman at Duke and Elkind at Oxford), resulting in the joint paper [1].

It is perhaps interesting that for Phragmén's rule and Monroe's rule moving from the optimization-based version of the rule to its sequential variant preserves JR, but for PAV this is not the case; moreover, compliance with JR cannot be restored by tweaking the weights. Thus, sequential variants of weighted approval rules are not faithful approximations of the original rules.

Now, let us revisit our example. Each of the rules that provide JR guarantees that there will be at least one formal methods talk. But we also want to ensure that at least two speakers representing the algorithms and complexity community are selected, and some of our rules do not have this property. For instance, depending on the distribution of approvals, SeqCCAV may select A, F, and two of the speakers supported by one logician each; this is not precluded by the JR axiom. Thus, perhaps we need a stronger axiom, which says that very large groups of voters with shared interests should get not one, but several representatives, in proportion to their size and cohesiveness.

To formalize this idea, it will be convenient to introduce additional terminology: for each $\ell \geq 1$ we say that a group of voters $V \subseteq N$ is *ℓ-large* if $|V| \geq \ell \cdot \frac{n}{k}$; V is *ℓ-cohesive* if $|\bigcap_{i \in V} A_i| \geq \ell$. We will now formulate two axioms that have been proposed in the literature; both axioms aim to capture the idea that large, cohesive groups deserve several representatives.

Extended Justified Representation (EJR). A committee W provides *extended justified representation (EJR)* if for every $\ell \in [k]$ and every ℓ-large, ℓ-cohesive group of voters V there exists a voter $i \in V$ who approves at least ℓ members of W, i.e., $|A_i \cap W| \geq \ell$.

Proportional Justified Representation (PJR). A committee W provides *proportional justified representation (PJR)* if for every $\ell \in [k]$ and every ℓ-large, ℓ-cohesive group of voters V there are at least ℓ members of W who are approved by a voter from V, i.e., $|W \cap (\cup_{i \in V} A_i)| \geq \ell$.

We say that a committee selection rule satisfies EJR (respectively, PJR) if all committees in the output of this rule provide EJR (respectively, PJR).

The EJR axiom has been proposed by Aziz et al. [1]; the PJR axiom was subsequently suggested in a workshop paper by Sánchez Fernández et al. [18], which was later extended to a AAAI-17 paper [16]. Both axioms are stronger than JR; indeed, JR corresponds to setting $\ell = 1$ in either of these axioms. Both EJR and PJR say that large, cohesive groups should get several representatives, but they interpret this requirement differently, with EJR being more demanding: if one voter gets at least ℓ representatives then clearly collectively the voters in V get at least ℓ representatives. An attractive feature of PJR is that it is compatible with the *perfect representation axiom*: in essence, this axiom says that if there is a committee that perfectly represents the electorate, in the sense that each committee member is approved by exactly $\frac{n}{k}$ voters, then some such committee should be selected. In contrast, EJR is not compatible with this axiom [18, 16].

It is not immediately obvious whether a committee satisfying EJR or PJR always exists. However, it turns out that this is indeed the case, and, in fact, some of our voting rules always produce such committees.

Of course, we cannot expect AV or SeqPAV to satisfy these axioms, as they fail the weaker JR axiom. The other rules on our list satisfy JR; but do they satisfy PJR or EJR? The first attempt to identify rules that satisfy PJR was not fully successful: Sánchez Fernández et al. [16] argue that MonAV and GrMonAV satisfy PJR as long as k divides n, but show that these rules may violate PJR if this condition is not satisfied. Moreover, these rules fail EJR even if k divides n [1]. The variants of Phragmén's rule (i.e., Phragmen, SeqPhragmen and ODH) all satisfy PJR, but not EJR [8, 17]; this is also the case for the EAR rule [6].

In contrast, the PAV rule satisfies not just PJR, but EJR [1]. The argument is not difficult: just as in the proof of Theorem 1 we consider a committee that fails EJR and identify a swap that increases the total PAV score. Perhaps more surprisingly, both PJR and EJR characterize PAV within the class of w-AV rules: every w-AV rule where w is not equivalent to the harmonic weight sequence fails PJR (and hence EJR) [1, 16]. Thus, the PJR/EJR axioms justify the choice of harmonic weights in the definition of PAV, which was made by Thiele back in 1895. Finally, we can now explain how the EJR-Exact class of rules got its name: this class of rules was designed with the explicit purpose of obtaining committees that provide EJR, and all rules in this class satisfy EJR (we note that Sánchez Fernández et al. [15] actually describe a broader family of rules; while all rules in this family satisfy PJR, only the EJR-Exact rules provide EJR).

4 Algorithms

So far, we have not paid much attention to the computational complexity of identifying representative committees. However, in practice algorithmic complexity may be an important consideration, especially in large elections where the size of the committee to be elected is large as well. Thus, it would be desirable to have a voting rule that satisfies a powerful justified representation axiom (ideally, EJR) and is polynomial-time computable. Another relevant question is deciding whether a given committee provides (proportional/extended) justified representation.

We start by considering the second question. One may think that, to verify whether a committee provides JR, one has to look at all groups of voters of size at least $\frac{n}{k}$, which is obviously computationally expensive. However, one can obtain an answer much more efficiently by focusing on *alternatives*: for each alternative not in the committee, we can check if there is a 'large' group of voters who approve that alternative, but do not approve anyone in the committee. This check can be performed in polynomial time. Regrettably, this approach does not extend to verifying PJR/EJR, as we would have to consider *groups* of alternatives. Indeed, it has been proved that deciding whether a given committee provides PJR or EJR is coNP-complete [1, 4].

Now, finding an efficiently computable rule that satisfies JR is not difficult: for instance, SeqCCAV fits the bill, and so do many other sequential rules discussed in this paper. For PJR the task is somewhat more challenging: the original workshop paper [18] that introduced the concept of PJR could not identify a single polynomial-time computable rule that satisfies this axiom, and the GrMonAV rule, while efficiently computable, is only guaranteed to satisfy PJR when the committee size k divides the number of voters n. However, essentially at the same time it was established that SeqPhragmen satisfies PJR [8], and so does ODH [17]. Subsequently, it was proved that several rules that are based on fractional vote transfers are efficiently computable and provide PJR; this includes, in particular, the EAR rule of Aziz and Lee [6] and the rules of Sánchez Fernández et al. [15]. Thus, there are many different approaches that allow us to find committees providing PJR quickly and efficiently.

In contrast, designing an efficient procedure for finding committees that provide EJR turned out to be more challenging. Indeed, even sophisticated sequential rules such as SeqPhragmen and ODH were shown to fail EJR. However, in the beginning of 2017 two groups of researchers [5, 20] independently made the same observation: the proof that committees with the maximum PAV score satisfy EJR also applies to committees with *almost* optimal score. This observation suggests a simple local search algorithm, which we call LS-PAV: we start with an arbitrary committee W, and look for a committee member $c \in W$ and a non-member $a \in C \setminus W$ such that swapping c with a increases W's PAV score *by a significant amount*. The condition that the change should be significant guarantees that we will converge in polynomial time, yet the resulting committee still satisfies EJR. Thus, committees providing EJR can be computed in polynomial time!

While the swap-based algorithm is very attractive from a theoretical perspective, convincing voters to use it may be difficult: they would see LS-PAV as trying to optimize a certain quantity, but then stopping before the optimal value is reached. Many voters would find that unappealing, especially if they can identify a committee that has a (slightly) higher PAV score than the selected committee. On the other hand, the EJR-Exact rules of Sánchez Fernández et al. [15], which were proposed at around the same time as both of the local swap algorithms, also admit a polynomial-time winner determination algorithm and provide EJR, while operating similarly to traditional voting rules, such as Single Transferable Vote. However, their weight transfer mechanism is not easy to explain, and the proof that these rules satisfy EJR is considerably more complex than for LS-PAV.

Recall that checking whether a given committee provides EJR/PJR is computationally hard: that is, somewhat counterintuitively, constructing a committee that provides EJR/PJR from scratch is easier than deciding if a given committee has this property.

Very recently, the three manuscripts that provide polynomial-time procedures for computing committees that provide EJR [5, 20, 15] as well as the manuscript that proves that it is coNP-complete to test whether a given committee provides PJR were combined into a paper that was accepted to AAAI-18 [2]. We would like to remark that this violates the pattern established by the previous two papers on this topic, each of which first appeared as an MPREF paper, and then got extended to a AAAI paper with a somewhat larger set of co-authors.

5 Conclusions

While the basic questions concerning JR, PJR and EJR have been answered, further questions remain. For instance, Aziz et al. [1] propose a notion of justified representation that is even stronger than EJR and is based on core stability in an associated cooperative game. At the moment, it remains open whether every input admits a committee that provides this form of justified representation; it is known that PAV may fail to output a committee with this property even when it exists. Another open question is whether EJR is compatible with other important axioms, such as, in particular, committee monotonicity (this axiom says that if a is in a winning committee of size k, it should be in a winning committee of size $k+1$). Indeed, the research surveyed in this paper can be seen as a part of the broader agenda of axiomatizing approval-based committee selection rules (see also [12]) and understanding their computational complexity.

References

1 H. Aziz, M. Brill, V. Conitzer, E. Elkind, R. Freeman, and T. Walsh. Justified representation in approval-based committee voting. *Social Choice and Welfare*, 48(2):461–485, 2017.

2 H. Aziz, E. Elkind, S. Huang, M. Lackner, L. Sánchez Fernández, and P. Skowron. On the complexity of extended and proportional justified representation. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-2018)*, 2018.

3 H. Aziz, S. Gaspers, J. Gudmundsson, S. Mackenzie, N. Mattei, and T. Walsh. Computational aspects of multi-winner approval voting. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2015)*, pages 107–115, 2015.

4 H. Aziz and S. Huang. Computational complexity of testing proportional justified representation. Technical Report arXiv:1612.06476, arXiv.org, 2016.

5 H. Aziz and S. Huang. A polynomial-time algorithm to achieve extended justified representation. Technical Report arXiv:1703.10415, arXiv.org, 2017.

6 H. Aziz and B. Lee. Achieving proportional representation via voting. Technical Report arXiv:1708.07580, arXiv.org, 2017.

7 H. Aziz and T. Walsh. Justified representation in approval-based committee voting. In *Proceedings of the 8th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-2014)*, 2014.

8 M. Brill, R. Freeman, S. Janson, and M. Lackner. Phragmén's voting methods and justified representation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-2017)*, pages 406–413, 2017.

9 E. Elkind, P. Faliszewski, P. Skowron, and A. Slinko. Properties of multiwinner voting rules. *Social Choice and Welfare*, 48(3):599–632, 2017.

10 S. Janson. Phragmén's and Thiele's election methods. Technical Report arXiv:1611.08826, arXiv.org, 2016.

11 D. Kilgour. Approval balloting for multi-winner elections. In J. Laslier and R. Sanver, editors, *Handbook on Approval Voting*, pages 105–124. Springer, 2010.

12 M. Lackner and P. Skowron. Consistent approval-based multi-winner rules. Technical Report arXiv:1704.02453, arXiv.org, apr 2017.

13 L. E. Phragmén. Sur une méthode nouvelle pour réaliser, dans les élections, la représentation proportionnelle des partis. *Öfversigt af Kongliga Vetenskaps-Akademiens Förhandlingar*, 51(3):133–137, 1894.

14 A. Procaccia, J. Rosenschein, and A. Zohar. On the complexity of achieving proportional representation. *Social Choice and Welfare*, 30(3):353–362, April 2008.

15 L. Sánchez Fernández, E. Elkind, and M. Lackner. Committees providing EJR can be computed efficiently. Technical Report arXiv:1704.00356, arXiv.org, 2017.

16 L. Sánchez Fernández, E. Elkind, M. Lackner, N. Fernández García, J. Arias Fisteus, P. Basanta Val, and P. Skowron. Proportional justified representation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-2017)*, pages 670–676, 2017.

17 L. Sánchez Fernández, N. Fernández García, and J. Arias Fisteus. Fully open extensions to the D'Hondt method. Technical Report arXiv:1609.05370, arXiv.org, 2016.

18 L. Sánchez Fernández, N. Fernández García, J. Arias Fisteus, and P. Basanta Val. Some notes on justified representation. In *Proceedings of the 10th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-2016)*, 2016.

19 P. Skowron, P. Faliszewski, and J. Lang. Finding a collective set of items: From proportional multirepresentation to group recommendation. *Artificial Intelligence*, 241:191–216, 2016.

20 P. Skowron, M. Lackner, E. Elkind, and L. Sánchez Fernández. Optimal average satisfaction and extended justified representation in polynomial time. Technical Report arXiv:1704.00293, arXiv.org, apr 2017.

21 T. N. Thiele. Om flerfoldsvalg. In *Oversigt over det Kongelige Danske Videnskabernes Selskabs Forhandlinger*, pages 415–441. 1895.

22 N. Tideman. The single transferable vote. *The Journal of Economic Perspectives*, 9(1):27–38, 1995.

A Markov Chain Theory Approach to Characterizing the Minimax Optimality of Stochastic Gradient Descent (for Least Squares)

Prateek Jain[1], Sham M. Kakade[*2], Rahul Kidambi[3], Praneeth Netrapalli[4], Venkata Krishna Pillutla[5], and Aaron Sidford[6]

1 **Microsoft Research, Bangalore, India**
praneeth@microsoft.com
2 **University of Washington, Seattle, WA, USA**
sham@cs.washington.edu
3 **University of Washington, Seattle, WA, USA**
rkidambi@uw.edu
4 **Microsoft Research, Bangalore, India**
praneeth@microsoft.com
5 **University of Washington, Seattle, WA, USA**
pillutla@cs.washington.edu
6 **Stanford University, Palo Alto, CA, USA**
sidford@stanford.edu

Abstract

This work provides a simplified proof of the statistical minimax optimality of (iterate averaged) stochastic gradient descent (SGD), for the special case of least squares. This result is obtained by analyzing SGD as a stochastic process and by sharply characterizing the stationary covariance matrix of this process. The finite rate optimality characterization captures the constant factors and addresses model mis-specification.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Stochastic Gradient Descent, Minimax Optimality, Least Squares Regression

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.2

Category Invited Paper

1 Introduction

Stochastic gradient descent is among the most commonly used practical algorithms for large scale stochastic optimization. The seminal result of [9, 8] formalized this effectiveness, showing that for certain (locally quadric) problems, asymptotically, stochastic gradient descent is statistically minimax optimal (provided the iterates are averaged). There are a number of more modern proofs [1, 3, 2, 5] of this fact, which provide finite rates of

* Sham Kakade acknowledges funding from the Washington Research Foundation Fund for Innovation in Data-Intensive Discovery.

convergence. Other recent algorithms also achieve the statistically optimal minimax rate, with finite convergence rates [4].

This work provides a short proof of this minimax optimality for SGD for the special case of least squares through a characterization of SGD as a stochastic process. The proof builds on ideas developed in [2, 5].

SGD for least squares. The expected square loss for $w \in \mathbb{R}^d$ over input-output pairs (x, y), where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$ are sampled from a distribution $\mathcal{D}$, is:

$$L(w) = \frac{1}{2}\,\mathbb{E}_{(x,y)\sim\mathcal{D}}[(y - w \cdot x)^2]$$

The optimal weight is denoted by:

$$w^* := \operatorname*{argmin}_w L(w)\,.$$

Assume the argmin in unique.

Stochastic gradient descent proceeds as follows: at each iteration t, using an i.i.d. sample $(x_t, y_t) \sim \mathcal{D}$, the update of w_t is:

$$w_t = w_{t-1} + \gamma(y_t - w_{t-1} \cdot x_t)x_t$$

where γ is a fixed stepsize.

Notation. For a symmetric positive definite matrix A and a vector x, define:

$$\|x\|_A^2 := x^\top A x.$$

For a symmetric matrix M, define the induced matrix norm under A as:

$$\|M\|_A := \max_{\|v\|=1} \frac{v^\top M v}{v^\top A v} = \|A^{-1/2} M A^{-1/2}\|\,.$$

The statistically optimal rate. Using n samples (and for large enough n), the minimax optimal rate is achieved by the maximum likelihood estimator (the MLE), or, equivalently, the empirical risk minimizer. Given n i.i.d. samples $\{(x_i, y_i)\}_{i=1}^n$, define

$$\widehat{w}_n^{\mathrm{MLE}} := \arg\min_w \frac{1}{n}\sum_{i=1}^n \frac{1}{2}\,(y_i - w \cdot x_i)^2$$

where $\widehat{w}_n^{\mathrm{MLE}}$ denotes the MLE estimator over the n samples.

This rate can be characterized as follows: define

$$\sigma^2_{\mathrm{MLE}} := \frac{1}{2}\mathbb{E}\left[(y - w^* x)^2 \|x\|^2_{H^{-1}}\right]\,,$$

and the (asymptotic) rate of the MLE is $\sigma^2_{\mathrm{MLE}}/n$ [7, 10]. Precisely,

$$\lim_{n\to\infty} \frac{\mathbb{E}[L(\widehat{w}_n^{\mathrm{MLE}})] - L(w^*)}{\sigma^2_{\mathrm{MLE}}/n} = 1,$$

The works of [9, 8] proved that a certain averaged stochastic gradient method achieves this minimax rate, in the limit.

For the case of additive noise models (i.e. the "well-specified" case), the assumption is that $y = w^* \cdot x + \eta$, with η being independent of x). Here, it is straightforward to see that:

$$\frac{\sigma^2_{\mathrm{MLE}}}{n} = \frac{1}{2}\frac{d\sigma^2}{n}.$$

The rate of $\sigma^2_{\mathrm{MLE}}/n$ is still minimax optimal even among mis-specified models, where the additive noise assumption may not hold [6, 7, 10].

Assumptions. Assume the fourth moment of x is finite. Denote the second moment matrix of x as

$$H := \mathbb{E}[xx^\top],$$

and suppose H is strictly positive definite with minimal eigenvalue:

$$\mu := \sigma_{\min}(H).$$

Define R^2 as the smallest value which satisfies:

$$\mathbb{E}[\|x\|^2 xx^\top] \preceq R^2 \mathbb{E}[xx^\top].$$

This implies $\mathrm{Tr}(H) = \mathbb{E}\|x\|^2 \leq R^2$.

2 Statistical Risk Bounds

Define:

$$\Sigma := \mathbb{E}[(y - w^* x)^2 xx^\top],$$

and so the optimal constant in the rate can be written as:

$$\sigma^2_{\mathrm{MLE}} = \frac{1}{2}\mathrm{Tr}(H^{-1}\Sigma) = \frac{1}{2}\mathbb{E}\left[(y - w^* x)^2 \|x\|^2_{H^{-1}}\right],$$

For the mis-specified case, it is helpful to define:

$$\rho_{\mathrm{misspec}} := \frac{d\|\Sigma\|_H}{\mathrm{Tr}(H^{-1}\Sigma)},$$

which can be viewed as a measure of how mis-specified the model is. Note if the model is well-specified, then $\rho_{\mathrm{misspec}} = 1$.

Denote the average iterate, averaged from iteration t to T, by:

$$\overline{w}_{t:T} := \frac{1}{T-t}\sum_{t'=t}^{T-1} w_{t'}.$$

▶ **Theorem 1.** *Suppose $\gamma < \frac{1}{R^2}$. The risk is bounded as:*

$$\mathbb{E}[L(\overline{w}_{t:T})] - L(w^*)$$
$$\leq \left(\sqrt{\frac{1}{2}\exp\left(-\gamma\mu t\right)R^2\|w_0 - w^*\|^2} + \sqrt{\left(1 + \frac{\gamma R^2}{1-\gamma R^2}\rho_{\mathrm{misspec}}\right)\frac{\sigma^2_{\mathrm{MLE}}}{T-t}}\right)^2.$$

The bias term (the first term) decays at a geometric rate (one can set $t = T/2$ or maintain multiple running averages if T is not known in advance). If $\gamma = 1/(2R^2)$ and the model is well-specified ($\rho_{\text{misspec}} = 1$), then the variance term is $2\sigma_{\text{MLE}}/\sqrt{T-t}$, and the rate of the bias contraction is μ/R^2. If the model is not well specified, then using a smaller stepsize of $\gamma = 1/(2\rho_{\text{misspec}}R^2)$, leads to the same minimax optimal rate (up to a constant factor of 2), albeit at a slower bias contraction rate. In the mis-specified case, an example in [5] shows that such a smaller stepsize is required in order to be within a constant factor of the minimax rate. An even smaller stepsize leads to a constant even closer to that of the optimal rate.

3 Analysis

The analysis first characterizes a bias/variance decomposition, where the variance is bounded in terms of properties of the stationary covariance of w_t. Then this asymptotic covariance matrix is analyzed.

Throughout assume:

$$\gamma < \frac{1}{R^2}\,.$$

3.1 The Bias-Variance Decomposition

The gradient at w^* in iteration t is:

$$\xi_t := -(y_t - w^* \cdot x_t)x_t\,,$$

which is a mean 0 quantity. Also define:

$$B_t := \mathrm{I} - x_t x_t^\top\,.$$

The update rule can be written as:

$$\begin{aligned} w_t - w^* &= w_{t-1} - w^* + \gamma(y_t - w_{t-1}\cdot x_t)x_t \\ &= (\mathrm{I} - \gamma x_t x_t^\top)(w_{t-1} - w^*) - \gamma\xi_t \\ &= B_t(w_{t-1} - w^*) - \gamma\xi_t\,. \end{aligned}$$

Roughly speaking, the above shows how the process on $w_t - w^*$ consists of a contraction along with an addition of a zero mean quantity.

From recursion,

$$w_t - w^* = B_t\cdots B_1(w_0 - w^*) - \gamma\left(\xi_t + B_t\xi_{t-1} + \cdots + B_t\cdots B_2\xi_1\right)\,.$$

This immediately leads to the following lemma.

▶ **Lemma 2.** *The error is bounded as:*

$$\mathbb{E}[L(\overline{w}_{t:T})] - L(w^*) \le \frac{1}{2}\left(\sqrt{\mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | \xi_0 = \cdots = \xi_T = 0]} + \sqrt{\mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | w_0 = w^*]}\right)^2\,,$$

where

$$\begin{aligned} \mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | \xi_0 = \cdots = \xi_T = 0] &= \mathbb{E}\|B_t\cdots B_1(w_0 - w^*)\|_H^2\,, \\ \mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | w_0 = w^*] &= \gamma^2\mathbb{E}\|\xi_t + B_t\xi_{t-1} + \cdots + B_t\cdots B_2\xi_1\|_H^2\,. \end{aligned}$$

The first term can be interpreted as the bias. $\mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | \xi_0 = \cdots = \xi_T = 0]$ is the risk in a process without additive noise; the conditioning is a little misleading and is meant to denote the error in a process without additive noise. The second term, when squared, gives rise to the variance; it is the error under a process driven solely by noise where $w_0 = w^*$.

Proof. First, for vector valued random variables u and v, the fact that $(\mathbb{E}u^\top Hv)^2 \le \mathbb{E}[\|u\|_H^2]\mathbb{E}[\|v\|_H^2]$ implies

$$\mathbb{E}\|u+v\|_H^2 \le \left(\sqrt{\mathbb{E}\|u\|_H^2} + \sqrt{\mathbb{E}\|v\|_H^2}\right)^2 .$$

To complete the proof of the lemma, note $\mathbb{E}L(w) - L(w^*) = \frac{1}{2}\mathbb{E}\|w - w^*\|_H^2$. ◀

Bias. The bias term is characterized as follows:

▶ **Lemma 3.** *For all t,*

$$\mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | \xi_0 = \cdots = \xi_T = 0] \le \exp(-\gamma\mu t)\|w_0 - w^*\|^2 .$$

Proof. Assume $\xi_t = 0$ for all t. Observe:

$$\begin{aligned}
\mathbb{E}\|w_t - w^*\|^2 &= \mathbb{E}\|w_{t-1} - w^*\|^2 - 2\gamma(w_{t-1} - w^*)^\top \mathbb{E}[xx^\top](w_{t-1} - w^*) \\
&\quad + \gamma^2 (w_{t-1} - w^*)^\top \mathbb{E}[\|x\|^2 xx^\top](w_{t-1} - w^*) \\
&\le \mathbb{E}\|w_{t-1} - w^*\|^2 - 2\gamma(w_{t-1} - w^*)^\top H(w_{t-1} - w^*) \\
&\quad + \gamma^2 R^2 (w_{t-1} - w^*)^\top H(w_{t-1} - w^*) \\
&\le \mathbb{E}\|w_{t-1} - w^*\|^2 - \gamma\mathbb{E}\|w_{t-1} - w^*\|_H^2 \\
&\le (1 - \gamma\mu)\mathbb{E}\|w_{t-1} - w^*\|^2 ,
\end{aligned}$$

which completes the proof. ◀

Variance. Now suppose $w_0 = w^*$. Define the covariance matrix:

$$C_t := \mathbb{E}[(w_t - w^*)(w_t - w^*)^\top | w_0 = w^*]$$

Using the recursion, $w_t - w^* = B_t(w_{t-1} - w^*) + \gamma\xi_t$,

$$C_{t+1} = C_t - \gamma HC_t - \gamma C_t H + \gamma^2 \mathbb{E}[(x^\top C_t x)xx^\top] + \gamma^2\Sigma \tag{1}$$

which follows from:

$$\mathbb{E}[(w_t - w^*)\xi_{t+1}^\top] = 0 , \text{ and } \mathbb{E}[(x_{t+1}x_{t+1}^\top)(w_t - w^*)\xi_{t+1}^\top] = 0$$

(these hold since $w_t - w^*$ is mean 0 and both x_{t+1} and ξ_{t+1} are independent of $w_t - w^*$).

▶ **Lemma 4.** *Suppose $w_0 = w^*$. There exists a unique C_∞ such that:*

$$0 = C_0 \preceq C_1 \preceq \cdots \preceq C_\infty$$

where C_∞ satisfies:

$$C_\infty = C_\infty - \gamma HC_\infty - \gamma C_\infty H + \gamma^2 \mathbb{E}[(x^\top C_\infty x)xx^\top] + \gamma^2\Sigma . \tag{2}$$

Proof. By recursion,

$$\begin{aligned} w_t - w^* &= B_t(w_{t-1} - w^*) + \gamma\xi_t \\ &= \gamma\left(\xi_t + B_t\xi_{t-1} + \cdots + B_t\cdots B_2\xi_1\right) . \end{aligned}$$

Using that ξ_t is mean zero and independent of $B_{t'}$ and $\xi_{t'}$ for $t < t'$,

$$C_t = \gamma^2\left(\mathbb{E}[\xi_t\xi_t^\top] + \mathbb{E}[B_t\xi_{t-1}\xi_{t-1}^\top B_t] + \cdots + \mathbb{E}[B_t\cdots B_2\xi_1\xi_1^\top B_2^\top\cdots B_t^\top]\right)$$

Now using that $\mathbb{E}[\xi_1\xi_1^\top] = \Sigma$ and that ξ_t and $B_{t'}$ are independent (for $t \neq t'$),

$$\begin{aligned} C_t &= \gamma^2\left(\Sigma + \mathbb{E}[B_2\Sigma B_2] + \cdots + \mathbb{E}[B_t\cdots B_2\Sigma B_2^\top\cdots B_t^\top]\right) \\ &= C_{t-1} + \gamma^2\mathbb{E}[B_t\cdots B_2\Sigma B_2^\top\cdots B_t^\top] \end{aligned}$$

which proves $C_{t-1} \preceq C_t$.

To prove the limit exists, it suffices to first argue the trace of C_t is uniformly bounded from above, for all t. By taking the trace of update rule, Equation 1, for C_t,

$$\mathrm{Tr}(C_{t+1}) = \mathrm{Tr}(C_t) - 2\gamma\mathrm{Tr}(HC_t) + \gamma^2\mathrm{Tr}(\mathbb{E}[(x^\top C_t x)xx^\top]) + \gamma^2\mathrm{Tr}(\Sigma) .$$

Observe:

$$\mathrm{Tr}(\mathbb{E}[(x^\top C_t x)xx^\top]) = \mathrm{Tr}(\mathbb{E}[(x^\top C_t x)\|x\|^2]) = \mathrm{Tr}(C_t\mathbb{E}[\|x\|^2xx^\top]) \leq R^2\mathrm{Tr}(C_tH) \tag{3}$$

and, using $\gamma \leq 1/R^2$,

$$\mathrm{Tr}(C_{t+1}) \leq \mathrm{Tr}(C_t) - \gamma\mathrm{Tr}(HC_t) + \gamma^2\mathrm{Tr}(\Sigma) \leq (1-\gamma\mu)\mathrm{Tr}(C_t) + \gamma^2\mathrm{Tr}(\Sigma) \leq \frac{\gamma\mathrm{Tr}(\Sigma)}{\mu} .$$

proving the uniform boundedness of the trace of C_t. Now, for any fixed v, the limit of $v^\top C_t v$ exists, by the monotone convergence theorem. From this, it follows that every entry of the matrix C_t converges. ◀

▶ **Lemma 5.** *Define:*

$$\overline{w}_T := \frac{1}{T}\sum_{t=0}^{T-1} w_t .$$

and so:

$$\frac{1}{2}\mathbb{E}[\|\overline{w}_T - w^*\|_H^2 | w_0 = w^*] \leq \frac{\mathrm{Tr}(C_\infty)}{\gamma T}$$

Proof. Note

$$\begin{aligned} &\mathbb{E}[(\overline{w}_T - w^*)(\overline{w}_T - w^*)^\top | w_0 = w^*] \\ &\quad = \frac{1}{T^2}\sum_{t=0}^{T-1}\sum_{t'=0}^{T-1}\mathbb{E}[(w_t - w^*)(w_{t'} - w^*)^\top | w_0 = w^*] \\ &\quad \preceq \frac{1}{T^2}\sum_{t=0}^{T-1}\sum_{t'=t}^{T-1}\Big(\mathbb{E}[(w_t - w^*)(w_{t'} - w^*)^\top | w_0 = w^*] + \\ &\qquad\qquad \mathbb{E}[(w_{t'} - w^*)(w_t - w^*)^\top | w_0 = w^*]\Big) , \end{aligned}$$

double counting the diagonal terms $\mathbb{E}[(w_t - w^*)(w_t - w^*)^\top | w_0 = w^*] \succeq 0$. For $t \le t'$, $\mathbb{E}[(w_{t'} - w^*)|w_0 = w^*] = (\mathrm{I} - \gamma H)^{t'-t}\mathbb{E}[(w_t - w^*)|w_0 = w^*]$. To see why, consider the recursion $w_t - w^* = (\mathrm{I} - \gamma x_t x_t^\top)(w_{t-1} - w^*) - \gamma\xi_t$ and take expectations to get $\mathbb{E}[w_t - w^*|w_0 = w^*] = (\mathrm{I} - \gamma H)\mathbb{E}[w_{t-1} - w^*|w_0 = w^*]$ since the sample x_t is is independent of the w_{t-1}. From this,

$$\mathbb{E}[(\overline{w}_T - w^*)(\overline{w}_T - w^*)^\top | w_0 = w^*] \preceq \frac{1}{T^2}\sum_{t=0}^{T-1}\sum_{\tau=0}^{T-t-1} (I - \gamma H)^\tau C_t + C_t(\mathrm{I} - \gamma H)^\tau \,,$$

and so,

$$\begin{aligned}\mathbb{E}[\|\overline{w}_T - w^*\|_H^2 | w_0 = w^*] &= \mathrm{Tr}\big(H\mathbb{E}[(\overline{w}_T - w^*)(\overline{w}_T - w^*)^\top | w_0 = w^*]\big) \\ &\le \frac{1}{T^2}\sum_{t=0}^{T-1}\sum_{\tau=0}^{T-t-1} \mathrm{Tr}\big(H(\mathrm{I} - \gamma H)^\tau C_t\big) + \mathrm{Tr}\big(C_t(\mathrm{I} - \gamma H)^\tau H\big)\,.\end{aligned}$$

Notice that $H(\mathrm{I} - \gamma H)^\tau = (\mathrm{I} - \gamma H)^\tau H$ for any non-negative integer τ. Since $H \succ 0$ and $I - \gamma H \succeq 0$, $H(\mathrm{I} - \gamma H)^\tau \succeq 0$ because the product of two commuting PSD matrices is PSD. Also note that for PSD matrices A, B, $\mathrm{Tr}AB \ge 0$. Hence,

$$\begin{aligned}\mathbb{E}[\|\overline{w}_T - w^*\|_H^2 | w_0 = w^*] &\le \frac{2}{T^2}\sum_{t=0}^{T-1}\sum_{\tau=0}^{\infty} \mathrm{Tr}\big(H(\mathrm{I} - \gamma H)^\tau C_t\big) \\ &= \frac{2}{T^2}\sum_{t=0}^{T-1} \mathrm{Tr}\big(H(\sum_{\tau=0}^{\infty}(\mathrm{I} - \gamma H)^\tau) C_t\big) \\ &= \frac{2}{T^2}\sum_{t=0}^{T-1} \mathrm{Tr}\big(H(\gamma H)^{-1} C_t\big) \qquad (*) \\ &= \frac{2}{\gamma T^2}\sum_{t=0}^{T-1} \mathrm{Tr}(C_t) \\ &\le \frac{2}{\gamma T}\cdot \mathrm{Tr}(C_\infty)\,,\end{aligned}$$

from lemma 4 where $(*)$ followed from

$$(\gamma H)^{-1} = (\mathrm{I} - (\mathrm{I} - \gamma H))^{-1} = \sum_{\tau=0}^{\infty}(\mathrm{I} - \gamma H)^\tau \,,$$

and the series converges because $\mathrm{I} - \gamma H \prec \mathrm{I}$. ◀

3.2 Stationary Distribution Analysis

Define two linear operators on symmetric matrices, $\mathcal{S}$ and $\mathcal{T}$ — where $\mathcal{S}$ and $\mathcal{T}$ can be viewed as matrices acting on $\binom{d+1}{2}$ dimensions — as follows:

$$\mathcal{S} \circ M := \mathbb{E}[(x^\top M x)xx^\top]\,, \qquad \mathcal{T} \circ M := HM + MH\,.$$

With this, C_∞ is the solution to:

$$\mathcal{T} \circ C_\infty = \gamma\mathcal{S} \circ C_\infty + \gamma\Sigma \tag{4}$$

(due to Equation 2).

▶ **Lemma 6.** *(Crude C_∞ bound) C_∞ is bounded as:*

$$C_\infty \preceq \frac{\gamma\|\Sigma\|_H}{1-\gamma R^2}\,\mathrm{I}\,.$$

Proof. Define one more linear operator as follows:

$$\widetilde{\mathcal{T}} \circ M := \mathcal{T} \circ M - \gamma HMH = HM + MH - \gamma HMH\,.$$

The inverse of this operator can be written as:

$$\widetilde{\mathcal{T}}^{-1} \circ M = \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma\widetilde{\mathcal{T}})^t \circ M = \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma H)^t M (\mathrm{I} - \gamma H)^t\,.$$

which exists since the sum converges due to that $0 \preceq \mathrm{I} - \gamma H \preceq \mathrm{I}$.

A few inequalities are helpful: If $0 \preceq M \preceq M'$, then

$$0 \preceq \widetilde{\mathcal{T}}^{-1} \circ M \preceq \widetilde{\mathcal{T}}^{-1} \circ M'\,, \tag{5}$$

since

$$\widetilde{\mathcal{T}}^{-1} \circ M = \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma H)^t M (\mathrm{I} - \gamma H)^t \preceq \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma H)^t M' (\mathrm{I} - \gamma H)^t = \widetilde{\mathcal{T}}^{-1} \circ M'\,,$$

(which follows since $0 \preceq \mathrm{I} - \gamma H$). Also, if $0 \preceq M \preceq M'$, then

$$0 \preceq \mathcal{S} \circ M \preceq \mathcal{S} \circ M'\,, \tag{6}$$

which implies:

$$0 \preceq \widetilde{\mathcal{T}}^{-1} \circ \mathcal{S} \circ M \preceq \widetilde{\mathcal{T}}^{-1} \circ \mathcal{S} \circ M'\,. \tag{7}$$

The following inequality is also of use:

$$\Sigma \preceq \|H^{-1/2}\Sigma H^{-1/2}\| H = \|\Sigma\|_H H\,.$$

By definition of $\widetilde{\mathcal{T}}$,

$$\widetilde{\mathcal{T}} \circ C_\infty = \gamma\mathcal{S} \circ C_\infty + \gamma\Sigma - \gamma H C_\infty H\,.$$

Using this and Equation 5,

$$\begin{aligned} C_\infty &= \gamma\widetilde{\mathcal{T}}^{-1} \circ \mathcal{S} \circ C_\infty + \gamma\widetilde{\mathcal{T}}^{-1} \circ \Sigma - \gamma\widetilde{\mathcal{T}}^{-1} \circ (HC_\infty H) \\ &\preceq \gamma\widetilde{\mathcal{T}}^{-1} \circ \mathcal{S} \circ C_\infty + \gamma\widetilde{\mathcal{T}}^{-1} \circ \Sigma \\ &\preceq \gamma\widetilde{\mathcal{T}}^{-1} \circ \mathcal{S} \circ C_\infty + \gamma\|\Sigma\|_H \widetilde{\mathcal{T}}^{-1} \circ H\,. \end{aligned}$$

Proceeding recursively by using Equation 7,

$$\begin{aligned} C_\infty &\preceq (\gamma\widetilde{\mathcal{T}}^{-1} \circ \mathcal{S})^2 \circ C_\infty + \gamma\|\Sigma\|_H (\gamma\widetilde{\mathcal{T}}^{-1} \circ \mathcal{S}) \circ \widetilde{\mathcal{T}}^{-1} \circ H + \gamma\|\Sigma\|_H \widetilde{\mathcal{T}}^{-1} \circ H \\ &\preceq \gamma\|\Sigma\|_H \sum_{t=0}^{\infty} (\gamma\widetilde{\mathcal{T}}^{-1} \circ \mathcal{S})^t \circ \widetilde{\mathcal{T}}^{-1} \circ H\,. \end{aligned}$$

Using

$$\mathcal{S} \circ \mathrm{I} \preceq R^2 H$$

and

$$\widetilde{\mathcal{T}}^{-1} \circ H = \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma H)^{2t} H = \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma 2H + \gamma^2 H)^t H \preceq \gamma \sum_{t=0}^{\infty} (\mathrm{I} - \gamma H)^t H = \gamma (\gamma H)^{-1} H = \mathrm{I}$$

leads to

$$C_\infty \preceq \gamma \|\Sigma\|_H \sum_{t=0}^{\infty} (\gamma R^2)^t \mathrm{I} = \frac{\gamma \|\Sigma\|_H}{1 - \gamma R^2} \mathrm{I}\,,$$

which completes the proof. ◀

▶ **Lemma 7.** *(Refined C_∞ bound) The $\mathrm{Tr}(C_\infty)$ is bounded as:*

$$\mathrm{Tr}(C_\infty) \leq \frac{\gamma}{2} \mathrm{Tr}(H^{-1}\Sigma) + \frac{1}{2} \frac{\gamma^2 R^2}{1 - \gamma R^2} d \|\Sigma\|_H$$

Proof. From Lemma 6 and Equation 6,

$$\mathcal{S} \circ C_\infty \preceq \frac{\gamma \|\Sigma\|_H}{1 - \gamma R^2} \mathcal{S} \circ \mathrm{I} \preceq \frac{\gamma R^2 \|\Sigma\|_H}{1 - \gamma R^2} H\,.$$

Also, from Equation 2, C_∞ satisfies:

$$HC_\infty + C_\infty H = \gamma \mathcal{S} \circ C_\infty + \gamma \Sigma\,.$$

Multiplying this by H^{-1} and taking the trace leads to:

$$\begin{aligned} \mathrm{Tr}(C_\infty) &= \frac{\gamma}{2} \mathrm{Tr}(H^{-1} \cdot (\mathcal{S} \circ C_\infty)) + \frac{\gamma}{2} \mathrm{Tr}(H^{-1}\Sigma) \\ &\leq \frac{1}{2} \frac{\gamma^2 R^2}{1 - \gamma R^2} \|\Sigma\|_H \mathrm{Tr}(H^{-1}H) + \frac{\gamma}{2} \mathrm{Tr}(H^{-1}\Sigma) \\ &= \frac{1}{2} \frac{\gamma^2 R^2}{1 - \gamma R^2} d \|\Sigma\|_H + \frac{\gamma}{2} \mathrm{Tr}(H^{-1}\Sigma) \end{aligned}$$

which completes the proof. ◀

3.3 Completing the proof of Theorem 1

Proof. The proof of the theorem is completed by applying the developed lemmas. For the bias term, using convexity leads to:

$$\begin{aligned} \frac{1}{2} \mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | \xi_0 = \cdots \xi_T = 0] &\leq \frac{1}{2} R^2 \mathbb{E}[\|\overline{w}_{t:T} - w^*\|^2 | \xi_0 = \cdots \xi_T = 0] \\ &\leq \frac{1}{2} \frac{R^2}{T - t} \sum_{t'=t}^{T-1} \mathbb{E}[\|w_{t'} - w^*\|^2 | \xi_0 = \cdots \xi_T = 0] \\ &\leq \frac{1}{2} \exp(-\gamma \mu t) R^2 \|w_0 - w^*\|^2\,. \end{aligned}$$

For the variance term, observe that

$$\frac{1}{2} \mathbb{E}[\|\overline{w}_{t:T} - w^*\|_H^2 | w_0 = w^*] \leq \frac{\mathrm{Tr}(C_\infty)}{\gamma(T - t)} \leq \frac{1}{T - t} \left(\frac{1}{2} \mathrm{Tr}(H^{-1}\Sigma) + \frac{1}{2} \frac{\gamma R^2}{1 - \gamma R^2} d \|\Sigma\|_H \right),$$

which completes the proof. ◀

References

1. Francis R. Bach. Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression. *Journal of Machine Learning Research (JMLR)*, volume 15, 2014.
2. Alexandre Défossez and Francis R. Bach. Averaged least-mean-squares: Bias-variance tradeoffs and optimal sampling distributions. In *AISTATS*, volume 38, 2015.
3. Aymeric Dieuleveut and Francis R. Bach. Non-parametric stochastic approximation with large step sizes. *The Annals of Statistics*, 2015.
4. Roy Frostig, Rong Ge, Sham M. Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass. In *COLT*, 2015.
5. Prateek Jain, Sham M. Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Parallelizing stochastic approximation through mini-batching and tail-averaging. *CoRR*, abs/1610.03774, 2016.
6. Harold J. Kushner and Dean S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.
7. Erich L. Lehmann and George Casella. *Theory of Point Estimation*. Springer Texts in Statistics. Springer, 1998.
8. Boris T. Polyak and Anatoli B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, volume 30, 1992.
9. David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. *Tech. Report, ORIE, Cornell University*, 1988.
10. Aad W. van der Vaart. *Asymptotic Statistics*. Cambridge University Publishers, 2000.

Automated Synthesis: a Distributed Viewpoint

Anca Muscholl

LaBRI, University of Bordeaux, France

Abstract

Distributed algorithms are inherently hard to get right, and a major challenge is to come up with automated techniques for error detection and recovery. The talk will survey recent results on the synthesis of distributed monitors and controllers.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.1.3 Concurrent Programming

Keywords and phrases Synthesis, distributed programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.3

Category Invited Paper

1 Overview

Distributed applications represent a significant part of our everyday life. Typically, our personal data are stored on remote distributed servers, data management relies on remote applications, data-intensive computations are performed on computer clusters, etc. Since distributed applications are deployed at increasingly large scale, they have to be reliable and robust, satisfying stringent correctness criteria. But distributed programs are hard to get right, and errors can be very subtle[1].

Formal methods, and in particular model-checking, can produce rigorous, automated reliability proofs for hardware and software systems. The area has always had special interest in distributed applications, for two reasons. First, distributed programs are error prone, because programmers have to consider all possible effects induced by different schedulings of events. Second, testing, which is widely used for certifying sequential programs, tends to have low coverage in the distributed setting, because bugs are usually difficult to reproduce: they may happen under very specific thread schedules, and the likelihood of taking such corner-case schedules may be very low. As a consequence, automated verification techniques represent a crucial support in the development of reliable distributed applications.

Many recent advances in formal methods are motivated by a substantial increase in deploying distributed applications like robot-assisted systems, cloud-based services, etc. However, formal verification of distributed programs is still extremely challenging. The first challenge is *scalability*: automated verification, such as model-checking and other traditional exploration techniques, can only handle small instances of concurrent programs, mostly because of the very large number of possible states and of the asynchronicity of concurrent executions. The second challenge is *parametrization*: distributed protocols are usually designed to work for an arbitrary number of concurrent processes, which means that verification is required for an arbitrary number of processes. The third challenge is related to

[1] A well-known quotation of Leslie Lamport says "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

contextual assumptions: distributed programs are often designed under specific assumptions about synchronicity, failure behaviors, etc., which are often either difficult to model or even hard to formalize at an informal level.

Runtime verification. In traditional *model-checking*, distributed programs and algorithms are modeled as a set of communicating finite-state processes, and the correctness of all possible executions is specified in some temporal logic. If at all doable, model-checking usually requires finite abstractions and clever heuristics, like partial-order reduction [9, 17, 21], in order to cope with the scalability problem. *Runtime verification* is an appealing alternative method to traditional exhaustive exploration, situated somewhere between testing and model-checking (see e.g. the surveys [12, 10] and the dedicated conference RV running for more than 15 years). Runtime verification is about monitoring program executions against formal specifications, and is a support for error detection. But designing distributed monitors from a given specification is significantly more difficult than for sequential programs, since for sake of feasibility, let alone efficiency, the monitoring information has to be computed by a distributed algorithm using only the communication means provided by the execution of the monitored program. The challenge here is to come up with algorithms constructing efficient monitors on various communication architectures, and with a reasonable computational overhead.

Synthesis. In *reactive synthesis* the goal is to automatically derive from some given specification a reactive program, that is, a program interacting continuously with its environment. The notion of reactive synthesis goes back to work by Church in the 60s [1], and it has given birth to a beautiful and rich theory of automata, logics and games of infinite duration (see e.g. [11, 20]). The games paradigm captures the idea of interaction between a program and its environment, and computing a winning strategy amounts to construct a program that behaves correctly in any possible situation. Nowadays the efforts to translate the theory into practical algorithms are considerable: a growing number of synthesis tools are available, and there is the dedicated annual tool competition Syntcomp@CAV.

The reactive synthesis of distributed programs is a particularly attractive problem because distributed programs are notoriously hard to get right. The problem has been considered already in the early 90s, starting with a model proposed by Pnueli and Rosner [18]: processes communicate via shared variables synchronously, according to a fixed communication architecture. Each process has a partial view about the global state, since its knowledge is limited to its input variables. The partial knowledge of processes has as consequence that the distributed synthesis problem is decidable only for very restricted communication architectures, without so-called information forks [5], basically for pipelines only [14].

2 Going distributed

The main distinguishing feature of runtime verification is that it is performed at runtime, by continuously checking program executions against formal specifications. This opens the possibility to use it as support for error diagnosis, and also to deploy correction mechanisms when an illegal behavior of the program is detected.

The traditional runtime verification approach is to construct a monitor from a given property. The monitor is then used to check e.g. the current execution of the system. In other words, the monitor reads the finite trace incrementally and it is supposed to notify if an error occurs. While in model-checking all executions of the system are considered (often

with emphasis on infinite executions) runtime verification deals with a single execution at a time and does not require complete knowledge about the program or system. In other terms, runtime verification can be applied on black-box systems for which no model is available. It also represents a lightweight method regarding complexity, since from a theoretical viewpoint monitoring single traces simply corresponds to the word problem. The main issue in runtime monitoring is the complexity of the monitor, i.e., its memory and computation time requirements, as a monitor runs in parallel with the system and should not slow it down too much.

Designing distributed monitors from a given specification is far more challenging than for sequential programs, as the monitoring information has to be computed by a distributed algorithm. A straightforward, but impractical, way to monitor a distributed program is to synchronize the relevant components and to inquire about their states. A much better way to do this is to build local monitors that deduce the required information by recording and exchanging suitable information using mainly the communication means provided by the execution of the monitored program. So the main point about distributed monitoring is to avoid adding synchronization in the program, since this usually impacts negatively on the overall performance [19].

A very successful example for the automatic generation of distributed monitors has its roots in the theory of Mazurkiewicz traces [15]. Within this theory, Zielonka's theorem [22] is a prime example of synthesis of distributed, finite-state monitors. A Zielonka automaton is in essence the parallel composition of finite-state processes that synchronize over shared actions. Many researchers contributed to simplify the construction and to improve its complexity. The most recent construction [7] produces deterministic distributed monitors of size that is exponential in the number of processes (and polynomial in the size of a DFA for the monitoring property). It is very challenging to try to adapt Zielonka's construction to models involving other types of synchronization. Generally speaking, constructing distributed monitors for specific architectures, and under specific conditions, like robustness under failures, is an important open problem.

Reactive synthesis. Reactive synthesis lays the ground for error recovery, once errors have been detected through monitoring. It can support for example the design of controllers that are in charge of taking appropriate recovery steps, depending on the failure cause. Zielonka automata turned out to be a promising alternative for reactive synthesis as well. The crucial difference compared to the model of Pnueli-Rosner is that the synchronization of processes in a Zielonka controller entails an exchange of information about the local knowledge. So although information is still partial, it is in some sense complete, according to the communication architecture. The decidability status of reactive distributed synthesis in this model is still open, but the problem is known to be decidable when the communication structure is acyclic [8, 16]. This result, together with some decidability results for restricted communication [6, 13], makes the Zielonka version of distributed synthesis more attractive than the one of Pnueli-Rosner.

Recently, Petri games were proposed as another formulation of the distributed reactive synthesis. These games are played on Petri nets, with places that are either controllable (belonging to the system) or uncontrollable (belonging to the environment). As for Zielonka automata, the decidability status of the synthesis problem is open. The synthesis problem was shown to be decidable whenever there is a single environment token, and a bounded number of system tokens [4]. Petri net games are an interesting alternative to Zielonka automata, but the precise relation between the two models remains to be explored.

Negotiation diagrams [2], a concurrency model inspired by workflow nets, can offer another fruitful setting for the synthesis problem. A negotiation diagram describes a distributed system with a fixed set of sequential processes. The diagram is composed of "atomic negotiations", each one involving some subset of processes. An atomic negotiation starts when all its participants are ready to engage in it, and concludes with the selection of one out of a fixed set of possible outcomes; for each participant process, the outcome determines which atomic negotiations the process is willing to engage in at the next step. In essence, deterministic negotiations are stateless Zielonka automata, and it turns out that their analysis is algorithmically much easier in some cases. For example, sound diagrams have polynomial-time algorithms for Mazurkiewicz-invariant static analysis problems [3]. Negotiations are very likely to be an attractive setting for distributed synthesis as well.

References

1 Alonzo Church. Logic, arithmetics, and automata. *Proceedings of the International Congress of Mathematicians*, 1962.

2 Javier Esparza and Jörg Desel. On negotiation as concurrency primitive. In *Proc. CONCUR'13*, volume 8052 of *LNCS*, pages 440–454. Springer, 2013.

3 Javier Esparza, Anca Muscholl, and Igor Walukiewicz. Static analysis of deterministic negotiations. In *Proc. LICS'17*, pages 1–12. IEEE Computer Society, 2017.

4 Bernd Finkbeiner and Ernst-Ruediger Olderog. Petri games: Synthesis of distributed systems with causal memory. In *GandALF'14*, EPTCS, pages 217–230, 2014.

5 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *LICS'05*, pages 321–330. IEEE Computer Society, 2005.

6 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 275–286. Springer, 2004.

7 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *ICALP'10*, volume 6199 of *LNCS*, pages 52–63. Springer, 2010.

8 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *ICALP'13*, volume 7966 of *LNCS*, pages 275–286. Springer, 2013.

9 Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.

10 Klaus Havelund and Giles Reger. Runtime verification logics a language design perspective. In *Models, Algorithms, Logics and Tools – Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday.*, volume 10460 of *LNCS*. Springer, 2017.

11 Orna Kupferman and Moshe Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245–263, 1999.

12 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

13 P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS'05*, volume 3821 of *LNCS*, pages 201–212. Springer, 2005.

14 P. Madhusudan and P.S. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.

15 Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

16 Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In *FSTTCS'14*, volume 29 of *LIPIcs*, pages 639–651. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

17 Doron A. Peled. All from one, one for all: on model checking using representatives. In *CAV'93*, LNCS, pages 409–423. Springer, 1993.

18 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS'90*, pages 746–757. IEEE Computer Society, 1990.

19 Koushik Sen, Abhay Vardhan, Gul Agha, and Grigore Rosu. Efficient decentralized monitoring of safety in distributed systems. In *International Conference on Software Engineering (ICSE'04))*, pages 418–427. IEEE Computer Society, 2004.

20 Wolfgang Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*. Springer, 2008.

21 Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, number 483 in LNCS, pages 491–515, 1991.

22 W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.

Matrix Estimation, Latent Variable Model and Collaborative Filtering*†

Devavrat Shah

Massachusetts Institute of Technology, Cambridge, USA
devavrat@mit.edu

Abstract

Estimating a matrix based on partial, noisy observations is prevalent in variety of modern applications with recommendation system being a prototypical example. The non-parametric latent variable model provides canonical representation for such matrix data when the underlying distribution satisfies "exchangeability" with graphons and stochastic block model being recent examples of interest. Collaborative filtering has been a successfully utilized heuristic in practice since the dawn of e- commerce. In this extended abstract, we will argue that collaborative filtering (and its variants) solve matrix estimation for a generic latent variable model with near optimal sample complexity.

1998 ACM Subject Classification Probability and Statistics, Non-parametric Statistics

Keywords and phrases Matrix Estimation, Graphon Estimation, Collaborative Filtering

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.4

Category Invited Paper

1 Introduction

We consider the question of sparse matrix estimation (or completion) with noisy observations. As a prototype for such a problem, consider a noisy observation of a social network where observed interactions are signals of true underlying connections. We might want to predict the probability that two users would choose to connect if recommended by the platform, e.g. LinkedIn. As a second example, consider a recommendation system where we observe movie ratings provided by users, and we may want to predict the probability distribution over ratings for specific movie-user pairs. A popular collaborative filtering approach suggests using "similarities" between pairs of users to estimate the probability of a connection being formed or a movie being liked. Traditionally, the similarities between pair of users in a social network is computed by comparing the set of their friends or in the context of movie recommendation, by comparing commonly rated movies. In the sparse setting, however most pairs of users have no common friends, or most pairs of users have no commonly rated movies; thus there is insufficient data to compute the traditional similarity metrics.

In this work, the primary interest is to understand how well does such a simple, intuitive approach to compute similarities between pair of users for matrix estimation work. In the

* This work was partially supported by DARPA project W911NF-16-1-0551 and NSF project CMMI-1462158 .

† This extended abstract of talk at FSTTCS 2017 is based on two works: Lee, Li, Shah and Song (2016) [27] and Borgs, Chayes, Lee and Shah (2017) [7].

process, we provide a way to extend the notion of the similarities utilized in practice by incorporating information within a larger radius neighborhood rather than restricting only to immediate neighbors. We establish that it achieves best-known sample complexity which matches well known, conjectured lower bound for a special instance of the generic problem, the mixed membership stochastic block model.

1.1 Model, Problem Statement

The question discussed above can be mathematically formulated as a matrix estimation problem. Let F be an $n \times n$ matrix which we would like to estimate, and let Z be a noisy signal of matrix F such that $\mathbb{E}[Z] = F$. The available data is denoted by $(\mathcal{E}, M)$, where $\mathcal{E} \subset [n] \times [n]$ denotes the subset of indices for which data is observed, and M is the $n \times n$ symmetric data matrix[1] where $M(u, v) = Z(u, v)$ for $(u, v) \in \mathcal{E}$, and $M(u, v) = 0$ for $(u, v) \notin \mathcal{E}$. We can equivalently represent the data with an undirected weighted graph $\mathcal{G}$ with vertex set $[n]$, edge set $\mathcal{E}$, and edge weights given by M. We shall use graph and matrix notations in an interchangeable manner. Given the data $(\mathcal{E}, M)$, we would like to estimate the original matrix F. We assume a uniform sampling model, where each entry is observed with probability p independently of all other entries.

We shall assume that each $u \in [n]$ is associated to a latent variable $\alpha_u \in \mathcal{X}_1$, which is drawn independently across indices $[n]$ as per distribution P_1 over a bounded compact space $\mathcal{X}_1$. We shall assume that the expected data matrix can be described by the latent function f, i.e. $F(u, v) = f(\alpha_u, \alpha_v)$, where $f : \mathcal{X}_1 \times \mathcal{X}_1 \to \mathbb{R}$ is a symmetric function. We note that such a structural assumption or the so-called Latent Variable Model is a canonical representation for exchangeable arrays as shown by Aldous and Hoover [5, 22, 6]. For each observation, we assume that $\mathbb{E}[Z(u, v)] = F(u, v)$, $Z(u, v)$ is bounded and $\{Z(u, v)\}_{1 \le u < v \le n}$ are independent conditioned on the node latent variables.

The goal is to find smallest p, as a function of n and structural properties of f, so that there exists an algorithm that can produce $\hat{F}$, an estimate of matrix F, so that the Mean-Squared-Error (MSE) between $\hat{F}$ and F, $\frac{1}{n^2} \sum_{u,v \in [n]} (\hat{F}(u, v) - F(u, v))^2$, converges to 0 as $n \to \infty$.

1.2 Related Works

The matrix estimation problem introduced above, as special cases, includes problems from different areas of literature: matrix completion popularized in the context of recommendation systems, graphon estimation arising from the asymptotic theory of graphs, and community detection using the stochastic block model or its generalization known as the mixed membership stochastic block model. We shall discuss key representative results here. We discuss the scaling of the sample complexity with respect to d (model complexity, usually rank) and n for polynomial time algorithms, including results for both mean squared error convergence, exact recovery in the noiseless setting, and convergence with high probability in the noisy setting.

In the context of matrix completion, there has been much progress under the low-rank assumption and additive noise model. Most theoretically founded methods are based on spectral decompositions or minimizing a loss function with respect to spectral constraints [23, 24, 14, 15, 32, 30, 18, 17, 16].

[1] The asymmetric variation of this question can be casted as symmetric version. See [7] for a detailed discussion.

Most of the results in matrix completion require additive noise models, which do not extend to setting when the observations are binary or quantized. The Universal Singular Value Thresholding (USVT) estimator [16] is able to handle general bounded noise, although it requires a few log factors more in its sample complexity compared to what is conjectured optimal and what our result achieves for low-rank matrices.

There is also a significant amount of literature which looks at the estimation problem when the data matrix is binary, also known as 1-bit matrix completion, stochastic block model (SBM) parameter estimation, or graphon estimation. The latter two terms are found within the context of community detection and network analysis, as the binary data matrix can alternatively be interpreted as the adjacency matrix of a graph – which are symmetric, by definition. Under the SBM, each vertex is associated to one of d community types, and the probability of an edge is a function of the community types of both endpoints. Estimating the $n \times n$ parameter matrix becomes an instance of matrix estimation. In SBM, the expected matrix is at most rank d due to its block structure. Precise thresholds for cluster detection (better than random) and estimation have been established by [1, 2, 3]. Our work, both algorithmically and technically, draws insight from this sequence of works, extending the analysis to a broader class of generative models through the design of an iterative algorithm, and improving the technical results with precise MSE bounds.

The mixed membership stochastic block model (MMSBM) allows each vertex to be associated to a length d vector, which represents its weighted membership in each of the d communities. The probability of an edge is a function of the weighted community memberships vectors of both endpoints, resulting in an expected matrix with rank at most d. Recent work by [33] provides an algorithm for weak detection for MMSBM with sample complexity d^2n, when the community membership vectors are sparse and evenly weighted. They provide partial results to support a conjecture that d^2n is a computational lower bound, separated by a gap of d from the information theoretic lower bound of dn. Our result achieves close to this conjectured lower bound, with a sample complexity of $\omega(d^5n)$ in order to guarantee consistency, which is much stronger than weak detection, in a much more generic setting.

Graphon estimation extends SBM and MMSBM to the generic Latent Variable Model where the probability of an edge can be any measurable function f of real-valued types (or latent variables) associated to each endpoint. Graphons were first defined as the limiting object of a sequence of large dense graphs [13, 19, 29], with recent work extending the theory to sparse graphs [11, 12, 10, 34]. In the graphon estimation problem, we would like to estimate the function f given an instance of a graph generated from the graphon associated to f.

[20, 25] provide minimax optimal rates for graphon estimation; however a majority of the proposed estimators are not computable in polynomial time, since they require optimizing over an exponentially large space (e.g. least squares or maximum likelihood) [35, 9, 8, 20, 25]. [9] provided a polynomial time method based on degree sorting in the special case when the expected degree function is monotonic. To our knowledge, existing positive results for sparse graphon estimation require either strong monotonicity assumptions [9], or rank constraints as assumed in the SBM, the 1-bit matrix completion, and in this work.

We call special attention to the similarity based methods which are able to bypass the rank constraints, relying instead on smoothness properties of the latent function f (e.g. Lipschitz) [36] as well as this work [27, 7]. They hinge upon computing similarities between rows or columns by comparing commonly observed entries. Similarity based methods, also known in the literature as collaborative filtering, have been successfully employed across many large scale industry applications (Netflix, Amazon, Youtube) due to its simplicity and scalability [21, 28, 26, 31]; however the theoretical results have been relatively sparse.

These recent results suggest that the practical success of these methods across a variety of applications may be due to its ability to capture local structure.

A key limitation of this approach that this work overcomes is ability to deal with sparsity. In particular, [36] works when $p = 1$ (or all data is observed). Our result requires for $p = \omega(n^{-1/2})$ for any Lipschitz function instead, and $p = \omega(d^5 n^{-1})$ when the underlying model is low-rank with rank being d.

2 Algorithm

The algorithm that we propose uses the concept of local approximation, first determining which datapoints are similar in value, and then computing neighborhood averages for the final estimate. All similarity-based collaborative filtering methods have the following basic format:

1. Compute distances between pairs of vertices, e.g.,

$$\texttt{dist}(u,a) \approx \int_{\mathcal{X}_1} (f(\alpha_u, t) - f(\alpha_a, t))^2 dP_1(t). \tag{1}$$

2. Form estimate by averaging over "nearby" datapoints,

$$\hat{F}(u,v) = \frac{1}{|\mathcal{E}_{uv}|} \sum_{(a,b)\in\mathcal{E}_{uv}} M(a,b), \tag{2}$$

where $\mathcal{E}_{uv} := \{(a,b) \in \mathcal{E} \ s.t. \ \texttt{dist}(u,a) < \xi(n), \texttt{dist}(v,b) < \xi(n)\}$.

We will choose the threshold $\xi(n)$ depending on `dist` in order to guarantee that it is small enough to drive the bias to zero, ensuring the included datapoints are close in value, yet large enough to reduce the variance, ensuring $|\mathcal{E}_{uv}|$ diverges. In what follows, we describe two methods to compute distances. The first method uses immediate neighbors to estimate distance while the second utilizes far-away neighbors to estimate distance. Therefore, the first method works well when we have denser sampling, $p = \omega(n^{-1/2})$ while the second works for much sparse regime.

2.1 Distance using Immediate Neighbors

Like classical collaborative filtering using in practice, we approximate the L_2 distance of (1) by using variants of the finite sample approximation,

$$\texttt{dist}_0(u,a) = \frac{1}{|\mathcal{O}_{ua}|} \sum_{y\in\mathcal{O}_{ua}} (F(u,y) - F(a,y))^2, \tag{3}$$

where $y \in \mathcal{O}_{ua}$ iff $(u,y) \in \mathcal{E}$ and $(a,y) \in \mathcal{E}$ [4, 36, 27]. This approach works well when $p = \omega(n^{-1/2})$. However, for much sparser setting (i.e. $p = o(n^{-1/2})$) with high probability, $\mathcal{O}_{ua} = \emptyset$ for almost all pairs (u,a), such that this distance cannot be computed. This requires us to utilize far-way neighbors.

2.2 Distance using Far-away Neighbors

Some Notations. We shall assume that f has finite spectrum with rank d when regarded as an integral operator, i.e. for any $\alpha_u, \alpha_v \in \mathcal{X}_1$,

$$f(\alpha_u, \alpha_v) = \sum_{k=1}^{d} \lambda_k q_k(\alpha_u) q_k(\alpha_v),$$

where $\lambda_k \in \mathbb{R}$ for $1 \le k \le d$, q_k are orthonormal ℓ_2 functions for $1 \le k \le d$ such that

$$\int_{\mathcal{X}_1} q_k(y)^2 dP_1(y) = 1 \text{ and } \int_{\mathcal{X}_1} q_k(y)q_h(y)dP_1(y) = 0 \text{ for } k \neq h.$$

We assume that there exists some B_q such that $\sup_{y\in[0,1]} |q_k(y)| \le B_q$ for all k.

Let Λ denote the $d \times d$ diagonal matrix with $\{\lambda_k\}_{k\in[d]}$ as the diagonal entries, and let Q denote the $d \times n$ matrix where $Q(k,u) = q_k(\alpha_u)$. Since Q is a random matrix depending on the sampled α, it is not guaranteed to be an orthonormal matrix (even though q_k are orthonormal functions). By definition, it follows that $F = Q^T\Lambda Q$. Let $d' \le d$ be the number of distinct valued eigenvalues amongst $\lambda_k, 1 \le k \le d$. Let $\tilde{\Lambda}$ denote the $d \times d'$ matrix where $\tilde{\Lambda}(a,b) = \lambda_a^{b-1}$.

Intuition. To that end, visualize the data via a graph with edge set $\mathcal{E}$, then (3) corresponds to comparing common neighbors of vertices u and a. A natural extension when u and a have no common neighbors, is to instead compare the r-hop neighbors of u and a, i.e. vertices y which are at distance exactly r from both u and a. We compare the product of weights along edges in the path from u to y and a to y respectively, which in expectation approximates

$$\begin{aligned}&\textstyle\int_{\mathcal{X}_1^{r-1}} f(\alpha_u, t_1)(\prod_{s=1}^{r-2} f(t_s, t_{s+1}))f(t_{r-1}, \alpha_y)d\prod_{i\in[r-1]} P_1(t_i)\\ &\quad= \textstyle\sum_k \lambda_k^r q_k(\alpha_u)q_k(\alpha_y)\\ &\quad= e_u^T Q^T \Lambda^r Q e_y. \end{aligned} \tag{4}$$

We choose a large enough r such that there are sufficiently many "common" vertices y which have paths to both u and a, guaranteeing that our distance can be computed from a sparse dataset.

Definition of Distance. We first expand local neighborhoods of radius r around each vertex. Let $\mathcal{S}_{u,s}$ denote the set of vertices which are at distance s from vertex u in the graph defined by edge set[2] $\mathcal{E}$. Specifically, $i \in \mathcal{S}_{u,s}$ if the shortest path in $\mathcal{G} = ([n], \mathcal{E})$ from u to i has a length of s. Let $\mathcal{B}_{u,s}$ denote the set of vertices which are at distance at most s from vertex u in the graph defined by $\mathcal{E}$, i.e. $\mathcal{B}_{u,s} = \cup_{t=1}^{s} \mathcal{S}_{u,t}$. Let $\mathcal{T}_u$ denote a breadth-first tree in $\mathcal{G}$ rooted at vertex u. The breadth-first property ensures that the length of the path from u to i within $\mathcal{T}_u$ is equal to the length of the shortest path from u to i in $\mathcal{G}$. If there is more than one valid breadth-first tree rooted at u, choose one uniformly at random. Let $N_{u,r} \in [0,1]^n$ denote the following vector with support on the boundary of the r-radius neighborhood of vertex u (we also call $N_{u,r}$ the neighborhood boundary):

$$N_{u,r}(i) = \begin{cases} \prod_{(a,b)\in \text{path}_{\mathcal{T}_u}(u,i)} M(a,b) & \text{if } i \in \mathcal{S}_{u,r},\\ 0 & \text{if } i \notin S_{u,r}, \end{cases}$$

where $\text{path}_{\mathcal{T}_u}(u,i)$ denotes the set of edges along the path from u to i in the tree $\mathcal{T}_u$. The sparsity of $N_{u,r}(i)$ is equal to $|\mathcal{S}_{u,r}|$, and the value of the coordinate $N_{u,r}(i)$ is equal to the product of weights along the path from u to i. Let $\tilde{N}_{u,r}$ denote the normalized neighborhood boundary such that $\tilde{N}_{u,r} = N_{u,r}/|\mathcal{S}_{u,r}|$. We will choose radius $r = \frac{6\ln(1/p)}{8\ln(pn)}$.

[2] For establishing correctness of algorithm, the edges are divided into three random subsets. See [7] for details. Here, to keep exposition simple, we will ignore this technical aspect. We conjecture that similar results hold for this variant of the algorithm as well.

1. For each pair (u, v), compute $\texttt{dist}_1(u, v)$ according to

$$\left(\tfrac{1-p}{p}\right)\left(\tilde{N}_{u,r} - \tilde{N}_{v,r}\right)^T M\left(\tilde{N}_{u,r+1} - \tilde{N}_{v,r+1}\right).$$

2. For each pair (u, v), compute distance according to

$$\texttt{dist}_2(u, v) = \textstyle\sum_{i\in[d']} z_i \Delta_{uv}(r, i),$$

where $\Delta_{uv}(r, i)$ is defined as

$$\left(\tfrac{1-p}{p}\right)\left(\tilde{N}_{u,r} - \tilde{N}_{v,r}\right)^T M\left(\tilde{N}_{u,r+i} - \tilde{N}_{v,r+i}\right),$$

and $z \in \mathbb{R}^{d'}$ is a vector that satisfies $\Lambda^{2r+2}\tilde{\Lambda}z = \Lambda^2\mathbf{1}$. z always exists and is unique because $\tilde{\Lambda}$ is a Vandermonde matrix (see below where both Λ and $\tilde{\Lambda}$ are defined) and $\Lambda^{-2r}\mathbf{1}$ lies within the span of its columns.

3 Main Results

We state results for both type of distances: using immediate neighbors and using far-away neighbors.

3.1 Distance Using Immediate Neighbors

The distance defined using immediate neighbors (cf. (3)) was analyzed in [27]. It proves that a similarity based collaborative filtering-style algorithm provides a consistent estimator for matrix completion under the additive noise model with generic function as long as the latent function is Lipschitz, not just low rank; however, it requires $\tilde{O}(n^{3/2})$ samples. We refer a reader to see [27] precise statement of the theorem.

3.2 Distance Using Immediate Neighbors

The distance defined using far-away neighbors (cf. $\texttt{dist}_1$ and $\texttt{dist}_2$) was analyzed in [7]. It establishes that the expected squared error of the estimate computed in (2) using $\texttt{dist}_1$ converges to zero with n for $p = \omega(n^{-1+\epsilon})$ for some $\epsilon > 0$, i.e. p must be polynomially larger than n^{-1}. On the other hand, the expected squared error of the estimate computed in (2) using $\texttt{dist}_2$ conveges to zero for $p = \omega(d^5 n^{-1})$.

It should be noted that computing $\texttt{dist}_1$ does not require knowledge of the spectrum of f. But, computing $\texttt{dist}_2$ requires full knowledge of the eigenvalues $(\lambda_1 \dots \lambda_d)$ to compute the vector z. It seems plausible that the technique employed by [2] could be used to design a modified algorithm which does not need to have prior knowledge of the spectrium. They achieve this for the stochastic block model case by bootstrapping the algorithm with a method which estimates the spectrum first and then computes pairwise distances with the estimated eigenvalues.

Acknowledgements. I want to thank my co-authors Christian Borgs (Microsoft Research New England), Jennifer Chayes (Microsoft Research New England), Christina Lee (MIT / Microsoft Research New England / Cornell), Yihua Li (MIT / Microsoft) and Dogyoon Song (MIT). The bulk of the work reported in this extended abstract of talk is part of PhD Thesis of Christina Lee.

References

1 Emmanuel Abbe and Colin Sandon. Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 670–688. IEEE, 2015.

2 Emmanuel Abbe and Colin Sandon. Recovering communities in the general stochastic block model without knowing the parameters. In *Advances in neural information processing systems*, 2015.

3 Emmanuel Abbe and Colin Sandon. Detection in the stochastic block model with multiple clusters: proof of the achievability conjectures, acyclic bp, and the information-computation gap. *Advances in neural information processing systems*, 2016.

4 Edo M Airoldi, Thiago B Costa, and Stanley H Chan. Stochastic blockmodel approximation of a graphon: Theory and consistent estimation. In *Advances in Neural Information Processing Systems*, pages 692–700, 2013.

5 D.J. Aldous. Representations for partially exchangeable arrays of random variables. *J. Multivariate Anal.*, 11:581–598, 1981.

6 Tim Austin. Exchangeable random arrays. *Technical Report, Notes for IAS workshop.*, 2012.

7 Christian Borgs, Jennifer Chayes, Christina E. Lee, and Devavrat Shah. Thy friend is my friend: Iterative collaborating filtering for sparse graphon estimation. In *Advances in Neural Information Processing Systems 30*, 2017.

8 Christian Borgs, Jennifer Chayes, and Adam Smith. Private graphon estimation for sparse graphs. In *Advances in Neural Information Processing Systems*, pages 1369–1377, 2015.

9 Christian Borgs, Jennifer T Chayes, Henry Cohn, and Shirshendu Ganguly. Consistent nonparametric estimation for heavy-tailed sparse graphs. *arXiv preprint arXiv:1508.06675*, 2015.

10 Christian Borgs, Jennifer T Chayes, Henry Cohn, and Nina Holden. Sparse exchangeable graphs and their limits via graphon processes. *arXiv preprint arXiv:1601.07134*, 2016.

11 Christian Borgs, Jennifer T Chayes, Henry Cohn, and Yufei Zhao. An L^p theory of sparse graph convergence I: limits, sparse random graph models, and power law distributions. *arXiv preprint arXiv:1401.2906*, 2014.

12 Christian Borgs, Jennifer T Chayes, Henry Cohn, and Yufei Zhao. An L^p theory of sparse graph convergence II: Ld convergence, quotients, and right convergence. *arXiv preprint arXiv:1408.0744*, 2014.

13 Christian Borgs, Jennifer T Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. Convergent sequences of dense graphs I: Subgraph frequencies, metric properties and testing. *Advances in Mathematics*, 219(6):1801–1851, 2008.

14 Emmanuel Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2009.

15 Emmanuel J Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

16 Sourav Chatterjee. Matrix estimation by universal singular value thresholding. *The Annals of Statistics*, 43(1):177–214, 2015.

17 Yudong Chen and Martin J Wainwright. Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees. *arXiv preprint arXiv:1509.03025*, 2015.

18 Mark A Davenport, Yaniv Plan, Ewout van den Berg, and Mary Wootters. 1-bit matrix completion. *Information and Inference*, 3(3):189–223, 2014.

19 Persi Diaconis and Svante Janson. Graph limits and exchangeable random graphs. *Rendiconti di Matematica*, VII(28):33–61, 2008.

20 Chao Gao, Yu Lu, and Harrison H Zhou. Rate-optimal graphon estimation. *The Annals of Statistics*, 43(6):2624–2652, 2015.

21 David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 1992.

22 D.N. Hoover. Row-column exchangeability and a generalized model for probability. In *Exchangeability in Probability and Statistics (Rome, 1981)*, pages 281–291, 1981.

23 Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. *IEEE Transactions on Information Theory*, 56(6):2980–2998, 2010.

24 Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11(Jul):2057–2078, 2010.

25 Olga Klopp, Alexandre B Tsybakov, and Nicolas Verzelen. Oracle inequalities for network models and sparse graphon estimation. *To appear in Annals of Statistics*, 2015.

26 Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer US, 2011.

27 Christina E. Lee, Yihua Li, Devavrat Shah, and Dogyoon Song. Blind regression: Nonparametric regression for latent variable models via collaborative filtering. In *Advances in Neural Information Processing Systems 29*, pages 2155–2163, 2016.

28 Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

29 László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Society Providence, 2012.

30 Sahand Negahban and Martin J Wainwright. Estimation of (near) low-rank matrices with noise and high-dimensional scaling. *The Annals of Statistics*, pages 1069–1097, 2011.

31 Xia Ning, Christian Desrosiers, and George Karypis. *Recommender Systems Handbook*, chapter A Comprehensive Survey of Neighborhood-Based Recommendation Methods, pages 37–76. Springer US, 2015.

32 Benjamin Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12(Dec):3413–3430, 2011.

33 David Steurer and Sam Hopkins. Bayesian estimation from few samples: community detection and related problems. 2017.

34 Victor Veitch and Daniel M Roy. The class of random graphs arising from exchangeable random measures. *arXiv preprint arXiv:1512.03099*, 2015.

35 Patrick J Wolfe and Sofia C Olhede. Nonparametric graphon estimation. *arXiv preprint arXiv:1309.5936*, 2013.

36 Yuan Zhang, Elizaveta Levina, and Ji Zhu. Estimating network edge probabilities by neighborhood smoothing. *arXiv preprint arXiv:1509.08588*, 2015.

Some Open Problems in Information-Theoretic Cryptography

Vinod Vaikuntanathan

MIT CSAIL, Cambridge, MA, USA
vinodv@csail.mit.edu

Abstract

Information-theoretic cryptography is full of open problems with a communication-complexity flavor. We will describe several such problems that arise in the study of private information retrieval, secure multi-party computation, secret sharing, private simultaneous messages (PSM) and conditional disclosure of secrets (CDS). In all these cases, there is a huge (exponential) gap between the best known upper and lower bounds. We will also describe the connections between these problems, some old and some new.

1998 ACM Subject Classification G.1.0 Mathematics of Computing

Keywords and phrases Cryptography, Information-Theoretic Security, Private Information Retrieval, Secret Sharing, Multiparty Computation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.5

Category Invited Paper

1 Introduction

Information-theoretic cryptography deals with problems of secure communication and secure computation *against computationally unbounded adversaries*. While much of cryptography relies on unproven computational assumptions (and in particular, provides only conditional security), information-theoretic cryptography provides absolute guarantees that are independent of any computational assumption. As such, in the field of information-theoretic cryptography, one could hope to gain a complete understanding of the landscape of secure communication and computation, namely, classify which tasks are possible and which are not, and precisely quantify the computational and communication cost of security.

Indeed, Shannon's celebrated work [33] gave us such a complete picture for *secure communication* against unbounded adversaries: namely that the one-time pad is essentially the best one can do. While several influential works extended the basic model of secure communication to leverage environmental noise [35] or quantum effects [6] to accomplish information-theoretically secure communication in a larger range of settings, Shannon's characterizations gave us a clean and satisfying answer to the basic question.

The situation in *secure computation* turns out to be very different. Broadly speaking, secure computation [36, 22, 5, 10] deals with settings where two or more parties wish to communicate and compute a joint function f on their private inputs while revealing nothing to each other except the output of the computation. The primary complexity measure of secure computation protocols that we care about is their *communication complexity*.

Secure computation is replete with primitives and settings where there is an exponential gap between the known upper and lower bounds on communication complexity. In the basic setting of information theoretically secure 3-party computation that we describe in more

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 5; pp. 5:1–5:7

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

detail below, the best protocols to compute an arbitrary function incur communication cost $2^{O(n)}$ where n is the total bit-length of the inputs of all parties, whereas the best lower bounds are linear in n. This leads us to ask:

What is the true communication overhead of secure computation?

Furthermore, all known approaches for information-theoretically secure computation incur a communication cost that is proportional to the computational cost of the function (in some computational model, say as a Boolean or arithmetic circuit). Thus, these approaches get stuck at the so-called *circuit-size barrier*. Yet another fundamental question in the foundations of information-theoretic cryptography is:

Does the communication cost of secure computation depend on the computational cost?

It is worth noting here that a simple counting argument establishes the *existence* of functions that require exponentially large circuits, but a similar statement is not known for the communication cost. That is, we do not even know whether there *exist* functions that require super-linear communication to securely compute.

In the rest of this extended abstract, we describe a number of objects of interest in information-theoretic cryptography – private information retrieval, secure multiparty computation, private simultaneous messages, conditional disclosure of secrets, and secret sharing – and the relations among them, as well as the open problems associated to these objects.

2 Information-Theoretic Primitives and their Problems

Private Information Retrieval (PIR). PIR is a protocol among one or more non-communicating servers each holding the same database D, thought of as an N-bit string, and a user holding an index $i \in [N]$. The user wishes to retrieve the i-th bit $D[i]$ from the server(s), without revealing any information about i. Clearly, the server(s) can rather inefficiently accomplish this by sending the entire string D to the user. The objective of PIR, then, is to achieve this goal while communicating (significantly) less than N bits. Such PIR schemes are deemed non-trivial.

Chor, Goldreich, Kushilevitz and Sudan [11], who first defined PIR, also showed that non-trivial *single-server* PIR schemes (with communication less than N bits) require computational assumptions. One line of research that resulted from this work showed several constructions of single-server PIR with decreasing communication complexity under various cryptographic assumptions [27, 9, 28, 7, 20, 19, 8], culminating in [8] that achieves the asymptotically optimal communication complexity of $O(\log N + \lambda)$ bits where λ is the cryptographic security parameter.

Chor, Goldreich, Kushilevitz and Sudan also proposed the natural setting of multi-server PIR where two or more *non-communicating* servers each holding the same database D interact with the client holding an index i. The client is guaranteed that its index is private as long as the servers do not collude with each other.

In the spirit of our questions, let us mention here that the best two-server PIR protocols have total communication complexity $2^{\tilde{O}(\sqrt{\log N})}$ [15] while the lower bound is "merely" $(5 - o(1)) \log N$ [31, 34]. We refer the reader to the excellent survey of Yekhanin and the bibliography maintained by Gasarch [37, 17] for pointers to the long line of work on information-theoretically secure multi-server PIR protocols. Curiously, such PIR schemes turn out to be equivalent in a precise sense to locally decodable codes, an object that does not refer to privacy at all [26].

Secure Multiparty Computation (MPC). In the setting of MPC, a collection of k parties wish to collaborate to compute a publicly known function f on their respective inputs $x_1, x_2, \ldots, x_k$ without leaking any information to each other except the output $f(x_1, \ldots, x_k)$. Such a protocol should be secure against collusions of t corrupted parties. It is well-known that 2-party secure computation of even simple functions requires computational assumptions even against a semi-honest adversary, thus for our purposes, the simplest setting to think about is the 3-party setting with a single corrupted party.

Canonical ways of achieving MPC go through some explicit computational representation of the function f either as a circuit [5, 10] or as a branching program [23]. Consequently, there are functions f for which such MPC protocols have communication exponential in the (total) input length simply because, by a counting argument, there are functions f which require exponentially large circuits (resp., branching programs). Must this be the case?

The lower bounds on the communication complexity of MPC are few and far between. To the best of our knowledge, the state of the art is the work of Data, Prabhakaran and Prabhakaran [13] who show a $1.5n - o(n)$ lower bound for 3-party secure computation (in the presence of a single corrupted party) where n is the total input length of all the parties. Since insecure computation can be achieved by communicating just n bits, this shows that *achieving security has its price*. However, here again, there is a large gap in communication between known protocols and lower bounds.

Interestingly, the two problems we just discussed, namely PIR and MPC, turn out to be equivalent to each other. A beautiful result of Ishai and Kushilevitz [24] tells us that any k-server PIR protocol gives us a $(k+1)$-party secure computation protocol tolerating a single corruption with nearly the same communication complexity, and vice versa. In particular, in the case of 3 parties, the PIR protocol of [15] gives us a 3-party computation protocol for arbitrary functions with communication $2^{\tilde{O}(\sqrt{n})}$ where n is the total input size.

Private Simultaneous Messages (PSM). Feige, Kilian and Naor [16] considered multiparty computation in a very structured and clean model called the private simultaneous messages (PSM) model (inspired by the simultaneous messages model of Babai, Kimmel and Lokam [2]). In the two-party PSM setting, Alice has an input x and Bob has input y, and they share a common random string which is unknown to the outside world. They send a single message each to a referee called Charlie. In turn, after receiving these messages, Charlie should be able to learn $f(x, y)$ (for a fixed, publicly known, function f) but nothing else about x or y. Since neither Bob nor Alice receive any additional information in the course of the protocol, they learn nothing about each other's input.

Again, there are large gaps between lower and upper bounds in this model. Feige, Kilian and Naor showed that there are functions that require $1.5n - o(n)$ bits of communication in this model (where n is the total input length of Alice and Bob). In a recent work, Beimel, Ishai, Kumaresan and Kushilevitz [4] showed that every function f has a PSM protocol with communication $2^{n/4}$ where n again is the total input length. Narrowing this gap is an important open problem.

Conditional Disclosure of Secrets (CDS). Two-party conditional disclosure of secrets (CDS) first defined by Gertner, Ishai, Kushilevitz and Malkin [21] is an important special case of PSM: two parties want to disclose a secret to a third party if and only if their respective inputs satisfy some fixed predicate ϕ. Concretely, Alice holds x, Bob holds y and in addition, they both hold a secret $m \in \{0, 1\}$ (along with some additional private randomness w). Charlie knows both x and y but not m; Alice and Bob want to disclose m to Charlie iff

$\phi(x, y) = 1$. How many bits do Alice and Bob need to communicate to Charlie?

This is a very simple and natural model where non-private computation requires very little communication (just a single bit), whereas the best upper bound for private computation is exponential. Indeed, in the non-private setting, Alice or Bob can send m to Charlie, upon which Charlie computes $\phi(x, y)$ and decides whether to output m or $\perp$. This trivial protocol with one-bit communication is not private because Charlie learns m even when the predicate is false. In contrast, in the private setting, we have a big gap between upper and lower-bounds. The best upper bound we have for CDS for general predicates ϕ requires that Alice and Bob each transmits $2^{\tilde{O}(\sqrt{n})}$ bits [29]. This upper bound works by translating a special type of PIR protocol into a CDS scheme. Indeed, the communication complexity of $2^{\tilde{O}(\sqrt{n})}$ is closely related to that of the Dvir-Gopi 2-server PIR scheme.

The best known lower bound is $\Omega(\log n)$ [18, 1] which is a double-exponential factor away from the upper bound! A central open problem is to narrow this gap; a concrete question in this direction is to improve the lower bound to $\Omega(n)$ even for a non-explicit function. On this note, we remark that [18] show an $\Omega(\sqrt{n})$ lower bound for the inner product predicate for special type of CDS protocols where Charlie's reconstruction algorithm is a linear function computed on the messages of Alice and Bob.

One could of course consider multi-party versions of both PSM and CDS. Recently, [30] showed a CDS protocol that achieves the same complexity of $2^{\tilde{O}(\sqrt{n})}$ even for arbitarily many parties.

Secret Sharing. The classical problem of secret sharing [32], more precisely non-threshold secret sharing [25], is closely related to multiparty CDS (see, e.g., [30] for more details on this connection). For general non-threshold secret-sharing schemes, the best upper bounds on the (individual) share size are exponential in the number of parties n, namely $2^{n-o(n)}$, whereas the best lower bounds are nearly linear [12], namely $\Omega(n/\log n)$ (see Beimel's survey [3] for more details).

In summary, there is a rich landscape of problems in information-theoretic cryptography, all closely related to secure computation, where there is a large gap between known upper and lower bounds on their communication complexity. Furthermore, there are non-trivial relations between all these problems. For example, MPC and PIR are equivalent modulo computational considerations [24]; PSM is a special type of MPC with a restricted communication pattern; multi-server PIR protocols of a special form give us CDS protocols with an equivalent communication complexity [29]; and multi-party CDS protocols are closely related to secret sharing. Despite recent progress [4, 14, 1, 29, 30], much remains to be uncovered in this world, eventually leading us to a better understanding of the question: *What is the true communication overhead of secure computation?*

References

1 Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. *IACR Cryptology ePrint Archive*, 2017:164, 2017. URL: `http://eprint.iacr.org/2017/164`.

2 László Babai, Peter G. Kimmel, and Satyanarayana V. Lokam. Simultaneous messages vs. communication. In *STACS*, pages 361–372, 1995.

3 Amos Beimel. Secret-sharing schemes: A survey. In *Coding and Cryptology – Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings*, pages 11–46, 2011.

4 Amos Beimel, Yuval Ishai, Ranjit Kumaresan, and Eyal Kushilevitz. On the cryptographic complexity of the worst functions. In *TCC*, pages 317–342, 2014.

5 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10, 1988.

6 C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, page 175, 1984.

7 Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005.

8 Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.12.

9 Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999. doi:10.1007/3-540-48910-X_28.

10 David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19, 1988.

11 Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998. doi:10.1145/293347.293350.

12 László Csirmaz. The size of a share must be large. *J. Cryptology*, 10(4):223–231, 1997.

13 Deepesh Data, Manoj Prabhakaran, and Vinod M. Prabhakaran. On the communication complexity of secure computation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014 – 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 2014. doi:10.1007/978-3-662-44381-1_12.

14 Deepesh Data, Vinod M. Prabhakaran, and Manoj M. Prabhakaran. Communication and randomness lower bounds for secure computation. *IEEE Trans. Information Theory*, 62(7):3901–3929, 2016. doi:10.1109/TIT.2016.2568207.

15 Zeev Dvir and Sivakanth Gopi. 2-server PIR with sub-polynomial communication. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 577–584, 2015.

16 Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563, 1994.

17 William Gasarch. Web page on private information retrieval. http://www.cs.umd.edu/~gasarch/TOPICS/pir/pir.html.

18 Romain Gay, Iordanis Kerenidis, and Hoeteck Wee. Communication complexity of conditional disclosure of secrets and attribute-based encryption. In *CRYPTO (II)*, pages 485–502, 2015.

19 Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*

2009, Bethesda, MD, USA, May 31 – June 2, 2009, pages 169–178. ACM, 2009. doi:10.1145/1536414.1536440.

20 Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer, 2005. doi:10.1007/11523468_65.

21 Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000. doi:10.1006/jcss.1999.1689.

22 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

23 Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304, 2000.

24 Yuval Ishai and Eyal Kushilevitz. On the hardness of information-theoretic multiparty computation. In *Advances in Cryptology – EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 439–455, 2004.

25 Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

26 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86, 2000.

27 Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS'97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646125.

28 Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005. doi:10.1007/11556992_23.

29 Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Conditional disclosure of secrets via non-linear reconstruction. In *Advances in Cryptology – CRYPTO 2017 – 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 758–790, 2017.

30 Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Towards breaking the exponential barrier for general secret sharing. Cryptology ePrint Archive, Report 2017/1062, 2017. https://eprint.iacr.org/2017/1062.

31 Eran Mann. Private access to distributed information. In *Master's thesis, Technion – Israel Institute of Technology*, 1998.

32 Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979. doi:10.1145/359168.359176.

33 C. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

34 Stephanie Wehner and Ronald de Wolf. Improved lower bounds for locally decodable codes and private information retrieval. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, pages 1424–1436, 2005.

35 A. D. Wyner. The wire-tap channel. *The Bell System Technical Journal*, 54(8):1355–1387, Oct 1975. doi:10.1002/j.1538-7305.1975.tb02040.x.

36 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

37 Sergey Yekhanin. *LDCs and PIRs*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007. URL: http://hdl.handle.net/1721.1/42242.

Backward Deterministic Büchi Automata on Infinite Words

Thomas Wilke

Department of Computer Science, Kiel University, Germany
thomas.wilke@email.uni-kiel.de

Abstract

This paper describes how backward deterministic Büchi automata are defined, what their main features are, and how they can be applied to solve problems in temporal logic.

1998 ACM Subject Classification automata over infinite objects, modal and temporal logics

Keywords and phrases finite automata, infinite words, determinism, backward automata, temporal logic, separated formulas

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.6

Category Invited Paper

1 Introduction

In their famous 1959 paper on finite-state automata [14] Rabin and Scott prove that

1. the reverse of a given non-deterministic finite-state automaton recognizes the reverse of the language recognized by the given automaton and
2. every non-deterministic finite-state automaton can be turned into an equivalent deterministic finite-state automaton.

From an automata-theoretic point of view, these results can be interpreted as follows. The class of regular languages can be defined by either one of the following finite-state automaton models: forward deterministic, forward non-deterministic, backward deterministic, backward non-deterministic.

For infinite words, more precisely, for ω-words, the situation is the same as long as powerful acceptance conditions (such as the parity, Rabin, or Muller condition) are used [8, 13, 10]. For the plain Büchi acceptance condition (or even the more flexible generalized Büchi condition), the situation is different: forward non-deterministic, backward deterministic, and backward non-deterministic automata can be used to define the class of regular ω-languages [2, 3], but forward deterministic automata are strictly weaker [7]. So, in some sense, if one wants a "complete" deterministic automaton model with a simple acceptance condition for regular ω-languages, the model of choice is backward deterministic Büchi automata.

This paper explains how backward deterministic Büchi automata are defined, what their main features are, and how they can be applied to solve problems in temporal logic.

2 Backward deterministic Büchi automata

Backward deterministic Büchi automata were introduced by Carton and Michel in [2, 3], where these automata are termed "complete unambiguous Büchi automata" or "CUBA", for short.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 6; pp. 6:1–6:9

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

A backward deterministic Büchi automaton is determined by

- a finite set of states S,
- an initial Büchi condition $B \subseteq S$,
- a final condition $I \subseteq S$, and
- a transition relation $\Delta \subseteq S \times A \times S$ which is reverse deterministic in the sense that there is a function $\delta\colon S \leftarrow A \times S$ such that $\Delta = \{(\delta(a,s),a,s) \mid a \in A, s \in S\}$.

In a backward deterministic generalized Büchi automaton the initial condition is $\mathcal{B} \subseteq 2^S$; the elements of $\mathcal{B}$ are called Büchi sets.

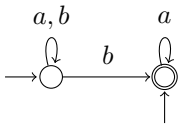
The above only describes the formal ingredients of a backward deterministic (generalized) Büchi automaton. There is also a semantic requirement that needs to be met. It is required that for every ω-word there is exactly one initial run. This means that for every ω-word $u \in A^\omega$ there is exactly one sequence $r \in S^\omega$ such that

- $s_i = \delta(a_i, s_{i+1})$ for every $i \in \omega$ and
- there exist infinitely many i such that $s_i \in B$.

For a generalized Büchi automaton, the second condition is replaced by:

- for every $B \in \mathcal{B}$, there exist infinitely many i such that $s_i \in B$.

Consider the language given by the ω-regular expression $(a+b)^* a^\omega$, which is described by the property "there are only finitely many occurrences of b". There is a simple non-deterministic Büchi automaton for this language:



The reverse of the transition relation is, in fact, a function, but there are no initial runs for the words which belong to the complement of the given language. In other words, this automaton must be extended in order to obtain a backward deterministic Büchi automaton for the language in question. Here is a simple way to do so:

Observe that the second, new component of this automaton has no final states, which is not surprising, because it is just there to make sure that for every word not (!) belonging to the language there is an initial run.

The requirement that there must be exactly one initial run on every word has an immediate important consequence: the automaton obtained from a given backward deterministic (generalized) Büchi automaton by complementing the initial condition (replace I by $S \setminus I$) is a backward deterministic (generalized) Büchi automaton which recognizes the complement of the language recognized by the given automaton:

▶ **Lemma 1** ([3]). *There are simple constructions that*

- *given a backward deterministic Büchi automaton, yield a backward deterministic Büchi automaton recognizing the complement of the language recognized by the given automaton,*
- *given two backward deterministic Büchi automata, yield a backward deterministic Büchi automaton recognizing the intersection (and union) of the language recognized by the given automata.*

The same is true for backward deterministic generalized Büchi automata.

The proof of the second claim is an instance of the question how to turn a backward deterministic generalized Büchi automaton into an equivalent backward deterministic automaton, because by a straightforward product construction one can, given two backward deterministic Büchi automata, construct a backward deterministic generalized Büchi automaton (with two Büchi sets) recognizing the intersection of the languages recognized by the given automata.

▶ **Theorem 2** ([3]). *For every backward deterministic generalized Büchi automaton with n states and m Büchi sets there exists an equivalent backward deterministic Büchi automaton with 2^{mn} states (even if transition conditions are used in the generalized Büchi condition, that is, if $\mathcal{B} \subseteq 2^{S \times A \times S}$).*

To conclude this section, we explain how the semantic property required of a backward deterministic ω-automaton can be checked. Let u be a non-empty finite word, say of length n, and $s \in S$ some state. Let $s_0 = s$ and $s_{i+1} = \delta(u(i), s_i)$ for i with $i < n$. Now u is said to be a loop at s if $s_n = s$ and $\{s_0, \ldots, s_n\}$ satisfies the initial condition, that is, $\{s_0, \ldots, s_n\} \cap B \neq \emptyset$ or $\{s_0, \ldots, s_n\} \cap B \neq \emptyset$ for every $B \in \mathcal{B}$.

▶ **Lemma 3** ([3]). *The semantic requirement (see above) is met if, and only if, every non-empty finite word is a loop at exactly one state. This can be checked in polynomial time.*

3 Completeness

The fundamental theorem on backward deterministic automata is:

▶ **Theorem 4** ([3]). *Every regular ω-language is recognized by a backward deterministic Büchi automaton. More precisely, for every ordinary (forward) non-deterministic Büchi automaton with n states there exists an equivalent backward deterministic Büchi automaton with at most $(12\,n)^n$ states.*

This theorem is due to Carton and Michel, as stated above. In their paper [3], Carton and Michel describe two different proofs. Another proof can be found in the book [11] by Perrin an Pin an yet another proof is given in [19].

The starting point for most of these proofs is an analysis of accepting runs (run DAGs) of ordinary Büchi automata.

4 Application to temporal logic

Backward deterministic ω-automata go very well with temporal logic—this is explained in the rest of this paper.

We consider both future-only linear-time temporal logic and future/past linear-time temporal logic. That is, we consider formulas which are built from propositional variables such as p and q using boolean operators and the temporal operators X (next), F (eventually), G (globally), U (until) for the future-only logic and these operators plus their past counterparts ($\overleftarrow{\mathsf{X}}$, $\overleftarrow{\mathsf{F}}$, $\overleftarrow{\mathsf{G}}$, and $\overleftarrow{\mathsf{U}}$) for the future/past logic.

A typical formula is

$$\mathsf{GF}p \ ,$$

which is to be read "always eventually p" (see above) and which means p holds infinitely often when the domain of time is, for instance, the canonical order of the natural numbers. In fact, in this paper we interpret temporal formulas in finite words or ω-words. The positions

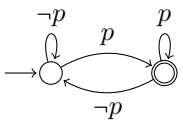
of such a word form the domain of time, the symbol at a position is a set of propositional (boolean) variables, exactly those variables that are true in the position. For instance, we have

$\{p\}\{q\}\{p\}\{q\}\ldots \vDash \mathsf{GF}p$, $\{q\}\{\}\{q\}\{\}\ldots \nvDash \mathsf{GF}p$.

4.1 From formulas to automata and back

There is a fundamental result on temporal logic and finite-state automata which says that a regular language of finite words is expressible in future-only or future/past temporal logic if, and only if, it is recognized by a counter-free automaton [6, 5, 9, 16]. Here, a state s is a mod-n counter for a non-empty finite word u in a given finite-state automaton with a transition function denoted δ if the states $s_0, s_1, \ldots, s_n$ defined by $\delta(s_i, u) = s_{i+1}$ for every $i < n$ are all distinct and $s_0 = s_n$ holds.

For example, consider the formula from above, which on finite words says that p holds in the last position, and a forward deterministic automaton for this language:



There is no counter in this automaton, because every word that leads from the left to the right state ends in $\{p\}$ and every word that leads from the right to the left state ends in $\{\}$.—Observe that we do not really attach symbols to the transitions in the diagram, but propositional formulas. Each such formula represents the variable assignments (which are the real symbols) that satisfy it.

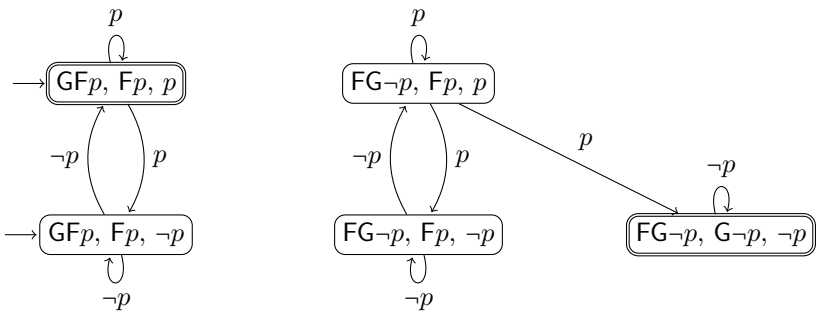
For proving the two directions of the above result it is easier to work with backward deterministic automata than with ordinary (forward) deterministic automata, or, alternatively, with past temporal formulas instead of future temporal formulas: the straightforward translation of future temporal formulas into non-deterministic automata [17], where the automaton guesses for each subformula where it is true in the given word and where it is not true, yields a backward deterministic automaton, even a backward deterministic counter-free automaton. For the other direction, from backward deterministic counter-free automata to future temporal formulas, a nested inductive argument can be used, see [18].

For ω-languages, that is, for future-only temporal logic on ω-words, similar techniques can be applied when working with backward deterministic (generalized) Büchi automata and a corresponding result holds:

▶ **Theorem 5** ([20]). *An ω-language is expressible in future temporal logic if, and only if, it is recognized by a counter-free backward deterministic (generalized) Büchi automaton.*

In the inductive proof of the more difficult direction, from automata to formulas, the corresponding result on finite words is used.

Here is the result of the straightforward translation of the formula $\mathsf{GF}p$ from above:



This automaton is, in fact, a backward deterministic Büchi automaton, and it is counter-free for similar reasons as in the previous example.—Note that the usual construction for turning a formula into an automaton results in a generalized Büchi condition with as many Büchi sets as there are F and U subformulas. So, here, we would expect two such sets, but these two are equivalent to the one given in the diagram.

4.2 Separated formulas

For future/past temporal formulas, there is no syntactic restriction on nesting future and past modalities. For instance,

$$\mathsf{F}(q \wedge \overleftarrow{\mathsf{X}}\,\overleftarrow{\mathsf{G}}\,p) \tag{1}$$

is, indeed, a formula. It is to be read "currently or sometime in the future q holds and p holds all the time before".

Here is a small diagram illustrating the property:

$$q\ q\ \ldots\ q\ q\ \boxed{q}\ q\ q\ \ldots\ q\ q\ p\ \ldots,$$

where the box denotes the present.

Obviously, the property can also be phrased "q holds all the time in the past and q holds until p holds sometime in the future or p holds currently". The corresponding formula is

$$\overleftarrow{\mathsf{G}}\,q \wedge q\mathsf{U}p\ . \tag{2}$$

This formula is such that past and future modalities are not nested. It is even true that the formula is a boolean combination of pure past, pure future and present (propositional) formulas.—When this is the case one says that a formula is separated.

A fundamental theorem by Gabbay states that every future/past temporal formula is equivalent to a separated one [4], see also [5]. In the following, it is explained how this result can be proved for the natural numbers using backward deterministic Büchi automata.

Unlike with future-only temporal formulas, which are usually interpreted in the first position of an ω-word, future/past formulas are interpreted in any position of an ω-word, as we have just seen. So the view that such a formula defines a set of ω-words is not appropriate anymore. Rather, it assigns to each position in an ω-word a truth value, namely whether or not the formula is true in that position. Formally, we can think of such a formula to define a function $A^\omega \to \{0,1\}^\omega$, where A is the underlying alphabet, for instance $2^{\{p,q\}}$.

From an automata-theoretic perspective, this requires a different automaton model. In fact, one would like to have an automaton model which describes, for every given ω-word

over the right alphabet, an ω-word over $\{0,1\}$, encoding where the formula in question is true, or, in other words, the value of the above function for this ω-word.

An appropriate automaton model one can work with is that of bimachine [15, 1]. A bimachine is given by

- a forward deterministic semi-automaton (no final state set) on finite words,
- a backward deterministic semi-automaton (no final state set) on ω-words, and
- a function $\lambda\colon Q \times A \times S \to \{0,1\}$.

Such a bimachine defines, indeed, a function $A^\omega \to \{0,1\}^\omega$. We describe how the value for some ω-word u is determined. Let us denote this value by w—recall that w is an ω-word over the alphabet $\{0,1\}$. Let $i \in \omega$ be any position. Then there is a unique state that the forward automaton assumes after having read the prefix $u(0)...u(i-1)$, say q. Similarly, there is a unique state that the backward deterministic semi-automaton assumes after having read the suffix $u(i+1)u(i+2)\ldots$, say s. Now, $w(i) = \lambda(q, u(i), s)$.

In other word, we let run the two automata from the start and the end of the word to the position in question and then combine the resulting states with the symbol of u at that position in order to determine the symbol of w at the same position:

$$q_0u(0)q_1u(1)q_2\ldots u(i-1)q_i \ , \qquad w(i) = \lambda(q_i, u(i), s_0) \ , \qquad s_0u(i+1)s_1u(i+2)\ldots \ .$$

The fundamental theorem now is:

▶ **Theorem 6** ([20]). *A function $A^\omega \to \{0,1\}^\omega$ is expressible in future/past temporal logic if, and only if, it is realized by a counter-free bimachine.*

Here, counter-free means that the forward and the backward automaton are counter-free.

The direction from left to right can be proved by an inductive argument where the inductive step involves an involved automata-theoretic construction.

The other direction follows from the fact that counter-free automata on finite words and backward deterministic counter-free Büchi automata define temporal properties. In fact, the prove yields immediately:

▶ **Corollary 7** ([4, 5]). *Every future/past temporal formula is equivalent to a separated formula.*

4.3 Effective characterizations of temporal logic and its fragments

The crucial ingredient of temporal logic are its temporal operators. Therefore, questions like "Can a given property be expressed

- without X?"
- with F only?"
- without nesting U?"
- by nesting U at most k times?"

are obvious questions to be asked. A similar question is: "Is a given regular ω-language definable in temporal logic at all?"—Backward deterministic ω-automata are a good means for designing corresponding decision procedures.

To describe appropriate decision criteria, we need some more preparation. We need to introduce a congruence relation on the state space of a given backward deterministic (generalized) Büchi automaton and to explain the notion "pattern".

Assume we are given a backward deterministic (generalized) Büchi automaton. Then this automaton has a deterministic transition function, say $\delta\colon S \leftarrow A \times S$ and a final state set $I \subseteq S$. We define an equivalence relation denoted $\equiv$ on S by: $s \equiv s'$ iff for every $u \in A^*$,

$\delta(u,s) \in I$ iff $\delta(u,s') \in I$. In a certain sense, this is a left congruence: whenever $s \equiv s'$, then $\delta(u,s) \equiv \delta(u,s')$ for every $u \in A^*$. Therefore, we can factor the transition function δ through $\equiv$. That is, a function $\delta_{\equiv}\colon S/{\equiv} \leftarrow A \times S/{\equiv}$ can be defined by

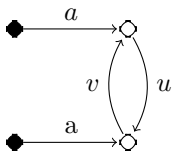
$$\delta_{\equiv}(a, s/{\equiv}) = \delta(a,s)/{\equiv} \quad . \tag{3}$$

An almost immediate consequence of Theorem 5 is the following characterization of temporal logic:

▶ **Theorem 8.** *An ω-language recognized by a backward deterministic (generalized) Büchi automaton with reverse transition function δ is definable in temporal logic if, and only if, the quotient transition function $\delta/{\equiv}$ does not have a counter.*

This answers the last question of the ones posed above. To show how the answer to one of the other question looks like, we consider F expressibility—the second question. We ask whether a given temporal property is expressible in the fragment of future temporal logic where F is the only temporal operator.

To describe an appropriate answer we need the notion "pattern". Here is an example of a pattern:



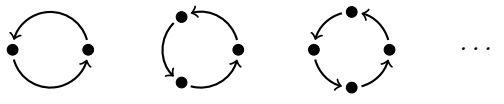
Such a graph is interpreted in the transition graph induced by a transition function δ. Symbols like a and b are variables for elements of the alphabet, symbols like u and v are variables for words over the alphabet, and full circles stand for distinct (!) states. So a backward deterministic automaton matches the above pattern if there are states s_0, s_1, s_2, and s_3 such that the following is true:

- there is a symbol $a \in A$ such that $s_3 = \delta(a, s_2)$ and $s_0 = \delta(a, s_1)$;
- there is some $u \in A^*$ such that $s_2 = \delta(u, s_1)$ and there is some $v \in A^*$ such that $s_1 = \delta(v, s_2)$;
- the states s_0 and s_3 are distinct.

Another way to put this is to say that there are states s_1 and s_2 in the same strongly connected component of the transition graph induced by δ and a symbol $a \in A$ such that $\delta(a, s_1) \neq \delta(a, s_2)$.

So Theorem 8 can be rephrased as follows.

▶ **Theorem 9** (Theorem 8 rephrased)**.** *An ω-language recognized by a backward deterministic (generalized) Büchi automaton with reverse transition function δ is definable in temporal logic if, and only if, the quotient transition function $\delta/{\equiv}$ does not have one of the following patterns.*

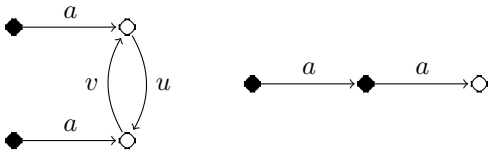


Recall the notion "loop" from above. In order to be able to state the criterion for F expressibility in a concise fashion, we need some more notation. When $u \in A^+$, we write $u\circlearrowright$ for the state at which u is a loop. We also need the notation $\mathrm{occ}(u)$ to denote the set of symbols occurring in a word u.

Now, the criterion for F expressibility is as follows.

▶ **Theorem 10** ([12]). *A temporal property is expressible in the* F *fragment of future temporal logic if, and only if, the following holds for any backward deterministic automaton for the ω-language defined by the property.*

1. *The quotient transition function $\delta/\equiv$ does not have one of the following patterns.*



2. *For all non-empty finite words u and v over the respective alphabet, if $u(0) = v(0)$ and $occ(u) = occ(v)$, then $u\wr = v\wr$.*

Observe that the second pattern states that the property is stutter-invariant: if an ω-word results from another ω-word by compressing or deflating chunks consisting of the same symbol, then either both words are accepted or neither one of them. This is an obvious requirement because the operator F is stutter-invariant.

5 Open problems

For several of the constructions with backward deterministic (generalized) Büchi automata, the exact complexity is not known. For instance, it would be good to have a reasonable upper bound for the translation of a future/past temporal formula into a counter-free bimachine.—Unlike with future-only temporal logic, the automata-theoretic constructions are involved. In fact, one step in the corresponding construction incurs an exponential blowup.

References

1 Olivier Carton. Right-sequential functions on infinite words. In Farid M. Ablayev and Ernst W. Mayr, editors, *Computer Science – Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings*, volume 6072 of *Lecture Notes in Computer Science*, pages 96–106. Springer, 2010. doi:10.1007/978-3-642-13182-0_9.

2 Olivier Carton and Max Michel. Unambiguous büchi automata. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 407–416. Springer, 2000. doi:10.1007/10719839_40.

3 Olivier Carton and Max Michel. Unambiguous büchi automata. *Theor. Comput. Sci.*, 297(1-3):37–81, 2003. doi:10.1016/S0304-3975(02)00618-7.

4 Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In Behnam Banieqbal, Howard Barringer, and Amir Pnueli, editors, *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer, 1987. doi:10.1007/3-540-51803-7_36.

5 Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal basis of fairness. In Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne, editors, *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*, pages 163–173. ACM Press, 1980. doi:10.1145/567446.567462.

6 Johan Anthony Willem Kamp. *Tense logic and the theory of linear order*. PhD thesis, University of California, Los Angeles, 1968.

7 Lawrence H. Landweber. Decision problems for omega-automata. *Mathematical Systems Theory*, 3(4):376–384, 1969. doi:10.1007/BF01691063.

8 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.

9 Robert McNaughton and Seymour Papert. *Counter-free automata*. M.I.T. Press research monographs. M.I.T. Press, 1971. URL: https://books.google.de/books?id=QRbvAAAAMAAJ.

10 Andrzej Wlodzimierz Mostowski. Regular expressions for infinite trees and a standard form of automata. In Andrzej Skowron, editor, *Computation Theory – Fifth Symposium, Zaborów, Poland, December 3-8, 1984, Proceedings*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 1984. doi:10.1007/3-540-16066-3_15.

11 Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Number 141 in Pure and Applied Mathematics. Elsevier, Amsterdam, 2004.

12 Sebastian Preugschat and Thomas Wilke. Effective characterizations of simple fragments of temporal logic using carton–michel automata. *Logical Methods in Computer Science*, 9(2), 2013. doi:10.2168/LMCS-9(2:8)2013.

13 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969. doi:10.2307/1995086.

14 Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.

15 Marcel Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4(2-3):185–196, 1961. doi:10.1016/S0019-9958(61)80006-5.

16 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

17 Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

18 Thomas Wilke. Classifying discrete temporal properties. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1999. doi:10.1007/3-540-49116-3_3.

19 Thomas Wilke. ω-automata. *CoRR*, abs/1609.03062, 2016. arXiv:1609.03062.

20 Thomas Wilke. Past, present, and infinite future. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 95:1–95:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.95.

Monitoring for Silent Actions*

Luca Aceto[1], Antonis Achilleos[2], Adrian Francalanza[3], and Anna Ingólfsdóttir[4]

1 **School of Computer Science, Reykjavik University, Reykjavik, Iceland and Gran Sasso Science Institute, L'Aquila, Italy**
2 **School of Computer Science, Reykjavik University, Reykjavik, Iceland**
3 **Dept. of Computer Science, ICT, University of Malta, Msida, Malta**
4 **School of Computer Science, Reykjavik University, Reykjavik, Iceland**

Abstract

Silent actions are an essential mechanism for system modelling and specification. They are used to abstractly report the occurrence of computation steps without divulging their precise details, thereby enabling the description of important aspects such as the branching structure of a system. Yet, their use rarely features in specification logics used in runtime verification. We study monitorability aspects of a branching-time logic that employs silent actions, identifying which formulas are monitorable for a number of instrumentation setups. We also consider defective instrumentation setups that imprecisely report silent events, and establish monitorability results for tolerating these imperfections.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Runtime Verification, Monitorability, Hennessy-Milner Logic with Recursion, Silent Actions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.7

1 Introduction

Runtime verification (RV) [19, 12] is a lightweight verification technique that strives to determine whether a system under scrutiny satisfies or violates a property – typically expressed as a formula from some logic – by incrementally analysing its current execution. In general, the runtime analysis is carried out by a *monitor*, a computational entity that observes the exhibited system execution and reaches a verdict once sufficient evidence is observed; the exhibited execution is characterised by a *trace*, a finite sequence of *events* describing the discrete system computational steps. Although the technique may obtain additional (runtime) information that could be useful for verification purposes, it is generally less expressive than exhaustive approaches such as model checking since the verification analysis is limited to the information inferred from the execution trace under consideration. *Monitorability* thus concerns itself with identifying the properties that are analysable by this runtime analysis.

RV setups typically partition computational steps of systems into two groups. On the one hand, *observable* events are those events that are visible (in full) to external entities such as monitors; they are used in the specifications describing system properties and are reported in the system trace. Observable events usually contain runtime data associated with that

* This research was supported by the project "TheoFoMon: Theoretical Foundations for Monitorability" (grant number: 163406-051) of the Icelandic Research Fund.

event (*e.g.* a method-call event would carry information relating the receiver, the method name and the arguments passed as parameters). On the other hand, *unobservable* events broadly encompass the computational steps that are abstracted away either from system modelling or from the respective property specifications; RV setups may occasionally remove these events from a trace so as to allow for a smoother monitoring process [15, 10].

In this work we investigate events that broadly fall somewhere in between these two groups. Concretely, *silent* events (or actions) are computational steps whose specific nature is not disclosed at the level of abstraction of the system model. Nevertheless the model still provides enough evidence of their manifestation during execution, which may play an important role in capturing vital behavioural aspects of the system: they may describe the branching structure of the modelled system behaviour [20, 16] or provide a measure of computational cost and efficiency [4]. In practice, one comes across various instances of such events. For example, the precise details of reported computational steps may be abstracted away for confidentiality/security reasons. Alternatively, the monitoring setup may be unable to report the details of certain computations due to limitations in the instrumentation technology used. In cyber-physical systems, there are also cases where one could detect the occurrence of certain (internal) computation by way of indirect means, such as via the sound of a running motor or the increase in temperature of an enclosed object. For these reasons, behavioural specifications often include descriptions involving silent actions. However, it is unclear how these silent actions are best handled in an RV setup. It is even less clear to what extent silent actions affect the monitorability of the respective specifications.

Our goals are to develop a foundational framework in which these questions may be addressed, and to logically characterize the properties that are monitorable within this framework. Following our work [13, 10, 1, 14, 2, 12, 11] and that of others [22, 7], we conduct our investigations in a process-calculus setting, where internal actions have long been studied from both behavioural and specification perspectives. Our study considers a standard labelled-transition-system model that represents silent computational steps as τ-transitions [20, 3], and a variant of the modal μ-calculus [17, 18] with *strong* modal operators that also describe τ-transitions. Our main contributions can be found in the middle sections of the paper:

- Section 3 studies the monitorability of this logic *w.r.t.* a number of monitoring setups that handle τ-actions differently, thus generalising the results obtained in [13, 14].
- Sections 4 and 5 investigate the monitorability of the logic for *imperfect* monitoring setups that obscure aspects of the silent system behaviour expressed by the model, and establish results for tolerating such imperfections.

2 Preliminaries

We assume the following disjoint sets: Act, a (possibly empty) set containing *external* actions, and Sil, a finite set containing *silent* actions. We let α range over Act, δ over Sil, and μ over $\text{Act} \cup \text{Sil}$. A *Labelled Transition System* (LTS) on (Act, Sil) is a triple

$$L = \langle P, (\text{Act}, \text{Sil}), \rightarrow_L \rangle,$$

where P is a nonempty set of system states referred to as *processes* $p, q, \ldots$, and $\rightarrow_L \subseteq P \times (\text{Act} \cup \text{Sil}) \times P$ is a transition relation. We write $p \xrightarrow{\mu}_L q$ instead of $(p, \mu, q) \in \rightarrow_L$ and $p \rightarrow_L q$ if $p \xrightarrow{\delta}_L q$ for some $\delta \in \text{Sil}$. We use $p \overset{\mu}{\Rightarrow}_L q$ to mean that, in L, p can derive q using a single μ action and any number of silent actions, that is, $p(\rightarrow_L)^*.\ \xrightarrow{\mu}_L .(\rightarrow_L)^* q$. We distinguish between (general) traces $s = \mu_1\mu_2\ldots\mu_r \in (\text{Act} \cup \text{Sil})^*$ and external traces

$t = \alpha_1\alpha_2\ldots\alpha_r \in \text{Act}^*$, and use $p \stackrel{s}{\Rightarrow}_L q$ to mean $p \stackrel{\mu_1}{\longrightarrow}_L . \stackrel{\mu_2}{\longrightarrow}_L \ldots \stackrel{\mu_r}{\longrightarrow}_L q$ and $p \stackrel{t}{\Rightarrow}_L q$ to mean $p \stackrel{\alpha_1}{\Longrightarrow}_L . \stackrel{\alpha_2}{\Longrightarrow}_L \ldots \stackrel{\alpha_r}{\Longrightarrow}_L q$. By $p \stackrel{\mu}{\longrightarrow}_L$ we mean that there is some q such that $p \stackrel{\mu}{\longrightarrow}_L q$. We occasionally omit the subscript L when it is clear from the context.

▶ **Example 1.** The (standard) regular fragment of CCS [20] with grammar:

$$p, q \in \text{Proc} ::= \quad \texttt{nil} \quad | \quad \mu.p \quad | \quad p + q \quad | \quad \texttt{rec}x.p \quad | \quad x,$$

with x from some countably infinite set of variables, and the transition relation defined as:

$$\text{Act}\frac{}{\mu.p \stackrel{\mu}{\longrightarrow} p} \qquad \text{Rec}\frac{p[\texttt{rec}x.p/x] \stackrel{\mu}{\longrightarrow} q}{\texttt{rec}x.p \stackrel{\mu}{\longrightarrow} q} \qquad \text{SelL}\frac{p \stackrel{\mu}{\longrightarrow} p'}{p + q \stackrel{\mu}{\longrightarrow} p'} \qquad \text{SelR}\frac{q \stackrel{\mu}{\longrightarrow} q'}{p + q \stackrel{\mu}{\longrightarrow} q'}$$

constitutes the LTS $\langle \text{Proc}, (\text{Act}, \{\tau\}), \rightarrow \rangle$ where τ is the only silent action.

Properties of processes may be specified via the logic μHML [18], a reformulation of the modal μ-calculus [17].

▶ **Definition 2.** μHML formulae on (Act, Sil) are defined by the grammar:

$$\begin{aligned}\varphi, \psi \in \mu\text{HML} ::= \quad & \texttt{tt} \quad | \quad \texttt{ff} \quad | \quad \varphi \wedge \psi \quad | \quad \varphi \vee \psi \\ & | \quad \langle\mu\rangle\varphi \quad | \quad [\mu]\varphi \quad | \quad \texttt{min}\ X.\varphi \quad | \quad \texttt{max}\ X.\varphi \quad | \quad X\end{aligned}$$

where X comes from a countably infinite set of logical variables LVar. For a given LTS $L = \langle P, (\text{Act}, \text{Sil}), \rightarrow \rangle$, an environment ρ is a function $\rho : \text{LVar} \to 2^P$. Given an environment ρ, $X \in \text{LVar}$, and $S \subseteq P$, $\rho[X \mapsto S]$ denotes the environment where $\rho[X \mapsto S](X) = S$ and $\rho[X \mapsto S](Y) = \rho(Y)$, for all $Y \neq X$. The semantics of a μHML formula φ over an LTS L relative to an environment ρ, denoted as $[\![\varphi, \rho]\!]_L$, is defined as follows:

$$\begin{aligned}[\![\texttt{tt}, \rho]\!]_L &= P \qquad [\![\texttt{ff}, \rho]\!]_L = \emptyset \qquad [\![X, \rho]\!]_L = \rho(X) \\ [\![\varphi_1 \wedge \varphi_2, \rho]\!]_L &= [\![\varphi_1, \rho]\!]_L \cap [\![\varphi_2, \rho]\!]_L \qquad [\![\varphi_1 \vee \varphi_2, \rho]\!]_L = [\![\varphi_1, \rho]\!]_L \cup [\![\varphi_2, \rho]\!]_L \\ [\![[\mu]\varphi, \rho]\!]_L &= \left\{ p \mid \forall q.\ p \stackrel{\mu}{\longrightarrow} q \text{ implies } q \in [\![\varphi, \rho]\!]_L \right\} \\ [\![\langle\mu\rangle\varphi, \rho]\!]_L &= \left\{ p \mid \exists q.\ p \stackrel{\mu}{\longrightarrow} q \text{ and } q \in [\![\varphi, \rho]\!]_L \right\} \\ [\![\texttt{min}\ X.\varphi, \rho]\!]_L &= \bigcap \{ S \mid S \supseteq [\![\varphi, \rho[X \mapsto S]]\!]_L \} \\ [\![\texttt{max}\ X.\varphi, \rho]\!]_L &= \bigcup \{ S \mid S \subseteq [\![\varphi, \rho[X \mapsto S]]\!]_L \}\end{aligned}$$

Two formulae φ and ψ are equivalent, denoted as $\varphi \equiv \psi$, when $[\![\varphi, \rho]\!]_L = [\![\psi, \rho]\!]_L$ for every environment ρ and LTS L. We often consider closed formulae and simply write $[\![\varphi]\!]_L$ for $[\![\varphi, \rho]\!]_L$, as their semantics is independent of ρ.

Let $[\text{Sil}]\varphi$ stand for $\bigwedge_{\delta \in \text{Sil}} [\delta]\varphi$ and $\langle \text{Sil} \rangle \varphi$ for $\bigvee_{\delta \in \text{Sil}} \langle\delta\rangle\varphi$. Then, the weak versions of the modalities employed in [13, 1, 14] may be expressed as follows:

$$[[\mu]]\varphi \equiv \texttt{max}\ X.([\mu]wb(\varphi) \wedge [\text{Sil}]X) \qquad \langle\langle\mu\rangle\rangle\varphi \equiv \texttt{min}\ X.(\langle\mu\rangle wd(\varphi) \vee \langle \text{Sil} \rangle X),$$

where $wb(\varphi) \equiv \texttt{max}\ Y.(\varphi \wedge [\tau]Y)$ and $wd(\varphi) \equiv \texttt{min}\ Y.(\varphi \vee \langle\tau\rangle Y)$. Readers should consult [18, 3], or more recently [14, 1], for more details on μHML.

Table 1 Behaviour and Instrumentation Rules for Monitored Systems

Monitor Semantics

$$\textsc{mRec}\ \frac{m[\mathtt{rec}x.m/x] \xrightarrow{\mu} m'}{\mathtt{rec}x.m \xrightarrow{\mu} m'} \qquad \textsc{mSel}\ \frac{m \xrightarrow{\mu} m'}{m + n \xrightarrow{\mu} m'} \qquad \textsc{mAct}\ \frac{}{\mu.m \xrightarrow{\mu} m} \qquad \textsc{mVrd}\ \frac{}{v \xrightarrow{\mu} v}$$

Instrumentation Semantics

$$\textsc{iMon}\ \frac{p \xrightarrow{\mu}_L q \quad m \xrightarrow{\mu}_M n}{m \triangleleft p \xrightarrow{\mu}_{I(M,L)} n \triangleleft q} \qquad \textsc{iTer}\ \frac{p \xrightarrow{\mu}_L q \quad m \overset{\mu}{\not\to}_M}{m \triangleleft p \xrightarrow{\mu}_{I(M,L)} \mathtt{end} \triangleleft q} \qquad \textsc{iAbs}\ \frac{p \xrightarrow{\delta}_L q}{m \triangleleft p \xrightarrow{\delta}_{I(M,L)} m \triangleleft q}$$

where $\mu \in \textsc{Act} \cup \textsc{Sil}$, $v \in \{\mathtt{end}, \mathtt{no}\}$, and $\delta \in \textsc{Sil}$.

3 Monitorability

The logic μHML of Section 2 is very expressive. It is also agnostic of the technique to be employed for verification. This level of generality provides an ideal basis for investigating the interplay between silent actions and the RV technique, and permits us to extend our findings to other specification logics (*e.g.* CTL and CTL* [8] can be encoded in μHML [17]). The property of monitorability, however, fundamentally relies on the monitoring setup considered.

Monitoring Systems. A *monitoring setup* on $(\textsc{Act}, \textsc{Sil})$ is a triple $S = \langle M, I, L\rangle$, where L is a system LTS on $(\textsc{Act}, \textsc{Sil})$, M is a monitor LTS on $(\textsc{Act}, \textsc{Sil})$, and I is the instrumentation describing how to compose L and M into an LTS, denoted by $I(M, L)$, on $(\textsc{Act}, \textsc{Sil})$. We call the pair (M, I) a *monitoring system* on $(\textsc{Act}, \textsc{Sil})$. For $M = (\textsc{Mon}, (\textsc{Act}, \textsc{Sil}), \rightarrow_M)$, $\textsc{Mon}$ is a set of monitor states (ranged over by m) and $\rightarrow_M$ is the *monitor semantics* described in terms of the behavioural state transitions a monitor takes when it analyses trace events $\mu \in \textsc{Act} \cup \textsc{Sil}$. The states of the composite LTS $I(M, L)$ are written as $m \triangleleft p$, where m is a monitor state and p is a system state; the monitored-system transition relation is here denoted by $\rightarrow_{I(M,L)}$. We focus on *rejection* monitors, *i.e.,* monitors with a designated rejection state $\mathtt{no}$, and hence safety fragments of the logic μHML. However, our arguments apply dually to acceptance monitors and co-safety properties; see [13, 14] for details.

▶ **Definition 3.** Fix a monitoring setup $S = \langle M, I, L\rangle$ on $(\textsc{Act}, \textsc{Sil})$ and let m be a monitor of M and φ a formula of μHML on $(\textsc{Act}, \textsc{Sil})$. We say that m (M, I)-*rejects* (or simply *rejects*, if M, I are evident) a process p in L, written as $\mathbf{rej}_S(m, p)$, when there are a process q in L and a trace $s \in (\textsc{Act} \cup \textsc{Sil})^*$ such that $m \triangleleft p \overset{s}{\Rightarrow}_{I(M,L)} \mathtt{no} \triangleleft q$. We say that m (M, I)-*monitors for* φ on L whenever

$$\text{for each process } p \text{ of } L,\ \mathbf{rej}_S(m, p) \text{ if and only if } p \notin [\![\varphi]\!]_L.$$

Finally, m (M, I)-*monitors for a formula* φ when m (M, I)-monitors for φ on L for every LTS L on $(\textsc{Act}, \textsc{Sil})$. The monitoring system (M, I) is often omitted when evident.

Monitoring for Silent Actions. The first monitoring system we consider does *not* distinguish between silent actions and external actions.

▶ **Definition 4.** A *full monitor* on $(\textsc{Act}, \textsc{Sil})$ is defined by the grammar:

$$m, n \in \textsc{Mon}_\delta ::= \quad \mathtt{end} \quad | \quad \mathtt{no} \quad | \quad \mu.m \quad | \quad m + n \quad | \quad \mathtt{rec}\ x.m \quad | \quad x,$$

where x comes from a countably infinite set of monitor variables. Constant `no` denotes the *rejection verdict* state whereas `end` denotes the *inconclusive verdict* state. The rules in Table 1 describe the behaviour for full monitors (we elide the obvious symmetric rule for $m + n$).

Note that rule mVrd in Table 1 describes how verdicts are irrevocable; monitors can therefore only describe suffix-closed behaviour.

▶ **Definition 5.** For any system LTS L and monitor LTS M agreeing on (Act, Sil), a *full instrumentation* LTS, denoted by $\rightarrow_{I(M,L)}$, is defined by rules iMon and iTer in Table 1.

In rule iMon, when the system produces a trace event μ that the monitor is able to analyse by transitioning from m to n, the constituent components of a monitored system $m \triangleleft p$ move in lockstep. Conversely, when the system produces an event μ that the monitor is *unable* to analyse, the monitored system still executes, according to iTer, but the monitor transitions to the inconclusive state, where it remains for the rest of the computation.

We refer to the monitor LTS in Definition 4 as M^δ, the full instrumentation of Definition 5 as I^δ and the pair (M^δ, I^δ) as the *full monitoring system*. For each system LTS L that agrees with the full monitoring system on (Act, Sil), we can show a correspondence between the respective monitoring setup $\langle M^\delta, I^\delta, L\rangle$ and the following syntactic subset of μHML.

▶ **Definition 6.** The *strong safety* μHML is defined by the grammar:

$$\theta, \chi \in \text{ssHML} ::= \quad \texttt{tt} \quad | \quad \texttt{ff} \quad | \quad [\mu]\theta \quad | \quad \theta \wedge \chi \quad | \quad \texttt{max}\ X.\theta \quad | \quad X$$

As opposed to sHML from [13, 14], ssHML is defined using strong transitions $p \xrightarrow{\mu} q$ (not weak ones, $p \stackrel{\mu}{\Rightarrow} q$) and the modalities $[\mu]\theta$ employ *any* action μ, not just external ones.

▶ **Definition 7.** Fix a monitoring system (M, I), a fragment Λ of μHML, and an LTS L on (Act, Sil). We say that (M, I) monitors for Λ on L whenever:

- For all $\varphi \in \Lambda$, there exists some $m \in M$ that monitors for it on L.
- For all $m \in M$, there exists some $\varphi \in \Lambda$ that is monitored by it on L.

We say that (M, I) monitors for Λ when it monitors for Λ on every LTS L.

▶ **Theorem 8.** *The* full monitoring system (M^δ, I^δ) *monitors for* ssHML.

Monitoring for External Actions. The results obtained in [13, 14] can be expressed and recovered within our more general framework.

▶ **Definition 9.** *Safety* μHML, presented in [13, 14], is defined by the grammar:

$$\pi, \kappa \in \text{sHML} ::= \quad \texttt{tt} \quad | \quad \texttt{ff} \quad | \quad [[\alpha]]\pi \quad | \quad \pi \wedge \kappa \quad | \quad \texttt{max}\ X.\pi \quad | \quad X.$$

Note that $[[\alpha]]\pi$ uses *external actions*. Its semantics is given as in Definition 2. We can also give a direct inductive definition, *i.e.*, $[\![[[\alpha]]\varphi, \rho]\!] = \{p \mid \forall q.\ p \stackrel{\alpha}{\Rightarrow} q \text{ implies } q \in [\![\varphi, \rho]\!]\}$.

▶ **Definition 10.** An *external monitor* on (Act, Sil) is defined by the grammar:

$$m, n \in \text{Mon}_\alpha ::= \quad \texttt{end} \quad | \quad \texttt{no} \quad | \quad \alpha.m \quad | \quad m + n \quad | \quad \texttt{rec}\ x.m \quad | \quad x.$$

Table 1 defines its LTS transition semantics, yielding $M^\alpha = \langle \text{Mon}_\alpha, (\text{Act}, \text{Sil}), \rightarrow\rangle$. *External instrumentation*, denote by I^α, is defined by the *three* rules iMon, iTer, and iAbs in Table 1; in the case of iMon and iTer action μ is substituted by the external action α. We refer to the pair (M^α, I^α) as the *external monitoring system*, amounting to the setup in [13, 14].

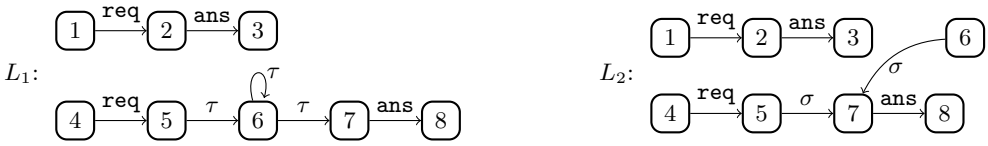


Figure 1 LTS L_1 depicts the two variations of the server from Example 12.

▶ **Theorem 11** (from [14]). *The* external monitoring system (M^α, I^α) *monitors for the sublogic* sHML.

▶ **Example 12.** Consider a simple server interface that receives requests from a client, represented by action `req`, and then sends a reply, represented by action `ans`. Between `req` and `ans`, a server implementation may upload a copy of the request transcript; this computation is represented as a sequence of silent τ-transitions that do not divulge information relating to the upload. In LTS L_1 of Figure 1, process 1 represents a server implementation that never uploads anything, whereas process 4 represents an alternative implementation that creates a transcript (the τ-transition from 5 to 6) and repeatedly attempts to upload the copy until it succeeds (the τ-loop on 6 followed by the transition to 7). An external monitor does not see processes 1 and 4 differently, as it does not observe the silent transitions. On the other hand, a full monitor can observe all the silent transitions that occur during an execution. We note that both process 1 and process 4 in L_1 violate the specification $[[\texttt{req}]][[\texttt{ans}]]\texttt{ff}$. Process 1 violates $[\texttt{req}][\texttt{ans}]\texttt{ff}$, while 4 does not. Conversely, process 1 does not violate the ssHML-specification $[\texttt{req}][[\tau]][\texttt{ans}]\texttt{ff}$, but 4 does: this can be observed by the full monitor $\texttt{req.rec}\ x.(\tau.(\texttt{ans.no} + x))$.

We conclude the section by commenting on other potential monitoring systems and their expressive power. In particular, the monitoring system (M^δ, I^α) yields monitoring setups whereby monitor δ-transitions are suppressed by the instrumentation, effectively making full monitors behave like external monitors from Definition 10. In the case of the monitoring system (M^α, I^δ), the instrumentation forces the monitor to transition to the inconclusive state more often since it does not abstract away from δ-transitions.

4 Obscuring the Silent Transitions

The full monitoring system (M^δ, I^δ) presented in Section 3 is straightforward and powerful. One might however argue that, in practice, it is *too* powerful: it is plausible that the visibility of certain silent transitions be somehow more *obscure* than that of external transitions. The external monitoring system (M^α, I^α) sits at the other end of the spectrum because it completely ignores all silent transitions. We consider monitoring systems that fall between these extremes: they can clearly observe certain silent transitions, but may receive imperfect information on others *i.e.,* observing that some number of transitions occurred, but *not* how many. In this case, we say that the transitions were *obscure*.

4.1 A Preorder on Obscure LTSs and Reliable Monitoring

We consider two silent actions: τ is a silent action that can be clearly observed and σ is the *obscure* silent action, representing an undetermined positive number of τ-transitions. In the following, we consider only monitoring setups on $(\text{Act}, \{\tau, \sigma\})$ and, whenever we say that L is an LTS, we mean that it is a system LTS on $(\text{Act}, \{\tau, \sigma\})$, unless otherwise stated; if L reports perfect information, it is assumed to be an LTS on $(\text{Act}, \{\tau\})$.

We consider a preorder $\leq_o$ on LTSs, where $L \leq_o L'$ intuitively means that L and L' have the same processes, but the silent transitions in L' are somehow more obscure than in L. Although we do not identify a specific such preorder, in Subsection 4.2, we introduce properties that we require of it. We say that L' is *an obscuring* of L when $L \leq_o L'$. We also introduce the obscuring preorder $\leq$ on $\text{Act} \cup \{\tau, \sigma\}$: $\mu_1 \leq \mu_2$ iff $\mu_1 = \mu_2$ or $\mu_1 = \tau$ and $\mu_2 = \sigma$. The intuition is that, whenever $\mu_1 \leq \mu_2$ and the system performs a μ_1-transition, the monitor may observe a (more obscure) μ_2-transition.

▶ **Example 13.** Consider a simple LTS L which contains exactly *one* maximal path:

$$p \xrightarrow{\mu_1}_L p_1 \xrightarrow{\mu_2}_L \cdots \xrightarrow{\mu_i}_L p_i \xrightarrow{\tau}_L q_1 \xrightarrow{\tau}_L \cdots \xrightarrow{\tau}_L q_r \xrightarrow{\mu_{i+1}}_L p_{i+1} \xrightarrow{\mu_{i+2}}_L \cdots \xrightarrow{\mu_k}_L p_k$$

of $k+r$ transitions, where $r > 0$; note that states $q_2 \dots q_{r-1}$ have no outgoing external actions. An obscuring of L may result from replacing $p_i \xrightarrow{\tau}_L q_1$ by a direct transition $p_i \xrightarrow{\sigma}_{L'} q_r$, thus obscuring the path $p_i \xrightarrow{\tau}_L q_1 \xrightarrow{\tau}_L \cdots \xrightarrow{\tau}_L q_r$ and leaving the remaining path unchanged. Thus in L' we have:

$$p \xrightarrow{\mu_1}_{L'} p_1 \xrightarrow{\mu_2}_{L'} \cdots \xrightarrow{\mu_i}_{L'} p_i \xrightarrow{\sigma}_{L'} q_r \xrightarrow{\mu_{i+1}}_{L'} p_{i+1} \xrightarrow{\mu_{i+2}}_{L'} \cdots \xrightarrow{\mu_k}_{L'} p_k$$

This would mean that as the system progresses from p to p_k, μ_1 through μ_k are clearly observed, but when the system performs $p_i \xrightarrow{\tau}_L q_1 \xrightarrow{\tau}_L \cdots \xrightarrow{\tau}_L q_r$, we only observe that at least one silent transition occurred, without discerning the exact number.

▶ **Definition 14.** Let m be a monitor of a monitoring system (M, I); φ a formula of μHML on $(\text{Act}, \{\tau\})$; L an LTS on $(\text{Act}, \{\tau\})$ – that is, L completely reports silent transitions; and L' an obscuring of L. We say that m (M, I)-*monitors* for φ on L from L' iff

$$\text{for every process } p \text{ of } L',\ p \notin [\![\varphi]\!]_L \text{ if and only if } \mathbf{rej}_{\langle M,I,L'\rangle}(m, p).$$

We say that m *reliably* (M, I)-*monitors* for φ on L if m (M, I)-monitors for φ on L from each obscuring of L. Monitor m reliably (M, I)-monitors for φ if m reliably (M, I)-monitors for φ on any LTS L. We often omit the monitoring system (M, I) whenever it is evident.

▶ **Definition 15.** Fix a monitoring system (M, I) and a fragment Λ of μHML on $(\text{Act}, \{\tau, \sigma\})$. (M, I) *reliably monitors* for Λ on LTS L iff

- For every $\varphi \in \Lambda$, there is a monitor m of M such that m reliably monitors for φ on L.
- For every monitor m of M, there is a $\varphi \in \Lambda$ such that m reliably monitors for φ on L.

(M, I) *reliably monitors* for Λ when (M, I) *reliably monitors* for Λ on every LTS.

4.2 Requirements on Obscuring Preorders

We identify certain properties of the obscuring ordering $\leq_o$ that we consider natural. These properties suffice to prove the results of Section 5. Consequently, the conclusions we draw about reliably monitorable formulas of μHML are proven for every $\leq_o$ that has these properties. Our intuition is that if $L \leq_o L'$, then L' is the same LTS as L, but seen with less precision with respect to the silent transitions. So, every transition we observe in L' is either a transition from L, or an obscure view of a sequence of transitions from L.

Natural Properties of Obscurings. We fix two LTSs $L \leq_o L'$. Since L' should at most provide imperfect information on the *silent* transitions of the system, external transitions should be unaffected:

A. $\xrightarrow{\alpha}_{L'} = \xrightarrow{\alpha}_L$ for every $\alpha \in \text{Act}$.

As L' obscures the information on the silent transitions of L, τ-transitions will become fewer: L' should have at most the τ-transitions of L (Property 4.2). Furthermore, every σ-transition in L' represents a non-empty sequence of silent transitions from L (Property 4.2).

B. $\xrightarrow{\tau}_{L'} \subseteq \xrightarrow{\tau}_{L}$ and

C. $\xrightarrow{\sigma}_{L'} \subseteq (\xrightarrow{\tau}_{L} \cup \xrightarrow{\sigma}_{L})^{+}$.

The following properties ensure that a certain level of information is retained in L'. In particular, if a state has a silent transition in L, it should still have a silent transition in L' (Property 4.2). Moreover, if a state p has a sequence of silent transitions in L that lead to a state q that can perform an external action, then this observation should be preserved in L'. Following Property 4.2, it suffices to require that q is reachable from p in L' via a sequence of silent actions (Property 4.2).

D. For all p if $p \xrightarrow{\tau}_{L}$ or $p \xrightarrow{\sigma}_{L}$, then $p \xrightarrow{\tau}_{L'}$ or $p \xrightarrow{\sigma}_{L'}$.

E. For all p, p', if $p(\xrightarrow{\tau}_{L} \cup \xrightarrow{\sigma}_{L})^{+}p' \xrightarrow{\alpha}$ for some $\alpha \in \text{Act}$, then $p(\xrightarrow{\tau}_{L'} \cup \xrightarrow{\sigma}_{L'})^{+}p'$.

The Strength of Obscuring. Properties 4.2, 4.2, and 4.2 capture the kind of obscuring ordering considered in this paper. We assume that there is a certain level of obscuring, beyond which adequate monitoring is deemed infeasible. In Property 4.2 below, obscuring can reach a point, represented as an LTS L^o, where all the silent-action information is hidden. That is, if $p \xrightarrow{\sigma}_{L^o} \xrightarrow{\sigma}_{L^o} p'$, then process p can also perform the more obscure transition $p \xrightarrow{\sigma}_{L^o} p'$ and furthermore, at no point does L^o reveal any clear τ-transition. We call such an obscuring L^o, as described by Property 4.2, a *total* obscuring.

F. Each L has an obscuring L^o, such that $\xrightarrow{\tau}_{L^o} = \emptyset$ and $\xrightarrow{\sigma}_{L^o}$ is transitive.

For an LTS L, let L^τ be the LTS on $(\text{Act}, \{\tau\})$ with the same set of processes, so that for $\alpha \in \text{Act}$, $\xrightarrow{\alpha}_{L^\tau} = \xrightarrow{\alpha}_{L}$ and $\xrightarrow{\tau}_{L^\tau} = \xrightarrow{\tau}_{L} \cup \xrightarrow{\sigma}_{L}$. Property 4.2 assures us that we can always obscure any selection of τ-transitions by turning them into σ-transitions, thus "forgetting" how many transitions were taking place at certain points. Property 4.2 can also be interpreted to mean that σ-transitions may indeed just represent single τ-transitions.

G. $L^\tau \leq_o L$ for each LTS L.

For the last requirement, we need the following definitions. For a process p in L and a trace $s \in (\text{Act} \cup \{\tau, \sigma\})^*$, we say that p *represents* s *in* L when s is the only maximal trace that p can produce – that is, when $\forall s'.\big(\exists q.\ p \overset{s'}{\Rightarrow}_{L} q$ iff s' is a prefix of $s\big)$. For a trace $s \in (\text{Act} \cup \{\sigma\})^*$, we define the *total obscuring* of s, denoted as $o(s)$, as follows: $o(\epsilon) = \epsilon$; $o(\sigma^k) = \sigma$ and $o(\sigma^k \alpha s) = \sigma\alpha\, o(s)$ for $k > 0$; and $o(\alpha s) = \alpha\, o(s)$. Property 4.2 ensures that any sequence of silent transitions can be obscured into a σ-transition at least for some LTSs:

H. for every trace $s \in (\text{Act} \cup \{\sigma\})^*$, there are LTSs $L \leq_o L'$ and a process p in L, such that p represents s in L and $o(s)$ in L'.

Property 4.2 may seem arbitrary, but it is not hard to justify that it is an immediate consequence of our intuition, as depicted in Example 13. Consider a maximal-path LTS L as in Example 13, but with τ-transitions replaced by σ-transitions, such that $s_1 = \mu_1 \cdots \mu_i \in \text{Act}^*$ and $s_2 = \mu_{i+1} \cdots \mu_k \in \text{Act}^*$. Then, p represents $s = s_1\sigma^k s_2$ in L and $o(s) = s_1\sigma s_2$ in L'.

4.3 An Ordering of Obscurings

We provide a natural instance of an ordering that has all the properties of Subsection 4.2.

▶ **Definition 16.** Relation $\leq_c$ is the transitive closure of $\leq_1$, where for LTSs L_1 and L_2 on $(\text{Act}, \{\tau, \sigma\})$, $L_1 \leq_1 L_2$ when for every $\alpha \in \text{Act}$, $\xrightarrow{\alpha}_{L_1} = \xrightarrow{\alpha}_{L_2}$ and one of the following holds:

1. $\xrightarrow{\tau}_{L_1} = \xrightarrow{\tau}_{L_2}$ and $\xrightarrow{\sigma}_{L_1} \subseteq \xrightarrow{\sigma}_{L_2} \subseteq \xrightarrow{\sigma}_{L_1} \cup \xrightarrow{\tau}_{L_1}$;

2. $\xrightarrow{\sigma}_{L_1} = \xrightarrow{\sigma}_{L_2}$ and $\xrightarrow{\tau}_{L_2} \subseteq \xrightarrow{\tau}_{L_1} \subseteq \xrightarrow{\sigma}_{L_2} \cup \xrightarrow{\tau}_{L_2}$;
3. $\xrightarrow{\tau}_{L_1} = \xrightarrow{\tau}_{L_2}$ and $\xrightarrow{\sigma}_{L_1} \subseteq \xrightarrow{\sigma}_{L_2} \subseteq (\xrightarrow{\sigma}_{L_1})^+$; or
4. $\xrightarrow{\tau}_{L_1} = \xrightarrow{\tau}_{L_2}$, $\xrightarrow{\sigma}_{L_2} \subseteq \xrightarrow{\sigma}_{L_1}$, and for all $p \xrightarrow{\sigma}_{L_1} p'$, if $p \not\xrightarrow{\sigma}_{L_2} p'$, then all the following hold:
   $p' \not\xrightarrow{\alpha}_{L_1}$ for all $\alpha \in \text{Act}$,
   $p' \xrightarrow{\sigma}_{L_1} p''$ or $p' \xrightarrow{\tau}_{L_1} p''$ for some $p'' \neq p'$, and
   $p \xrightarrow{\sigma}_{L_2} p''$ for every p'' such that $p' \xrightarrow{\sigma}_{L_1} p''$ or $p' \xrightarrow{\tau}_{L_1} p''$.

The cases presented in Definition 16 give a set of *moves* we can apply to construct a more obscure LTS from a given one. Informally:

1. According to move 1, for any transition $p \xrightarrow{\tau} q$, we can add transition $p \xrightarrow{\sigma} q$.
2. Following move 1, we can remove transition $p \xrightarrow{\tau} q$.
3. For transitions $p \xrightarrow{\sigma} p' \xrightarrow{\sigma} p''$, we can insert a new transition $p \xrightarrow{\sigma} p''$.
4. For transition $p \xrightarrow{\sigma} p'$, if move 3 has already been applied to $p \xrightarrow{\sigma} p' \xrightarrow{\sigma} p''$ for all possible and at least one $p' \xrightarrow{\delta} p''$, where $p' \neq p''$ and $\delta \in \{\tau, \sigma\}$, and $p' \not\xrightarrow{\alpha}$ for all $\alpha \in \text{Act}$ then we can remove $p \xrightarrow{\sigma} p'$.

▶ **Example 17.** We revisit Example 12 of a simple server. The LTS L_2 of Figure 1 presents a maximal obscuring of L_1, according to $\leq_c$. Moves 1 and 2 can replace all τ-transitions by σ-transitions; move 3 can be used to introduce a transition from process 5 directly to process 7; and move 4 can eliminate incoming transitions to process 6, including the self-loop. Thus, the LTS retains the information that the server uploads the transcript to a remote location, but not any information of intermediate steps. We observe that formula $\psi = [\texttt{req}][[\tau]][\texttt{ans}]\texttt{ff}$ is reliably monitorable on L_1 from L_2 by the full monitor $\texttt{req}.\texttt{rec}\ x.(\tau.(\texttt{ans}.\texttt{no} + x) + \sigma.(\texttt{ans}.\texttt{no} + x))$.

▶ **Proposition 18.** *Relation $\leq_c$ has all the properties listed in Subsection 4.2.*

5 Reliable Monitorability

In this section, we identify a maximal reliably monitorable fragment of μHML – up to logical equivalence – and a monitoring system that monitors for it. The results of this section are relative to any fixed preorder $\leq_o$ that satisfies the properties presented in Subsection 4.2.

▶ **Example 19.** Let $\varphi_1 = [\tau][\alpha]\texttt{ff}$ (*i.e.,* after any τ-action, a process cannot perform an α-action), $\varphi_2 = [\tau]\texttt{ff}$ (*i.e.,* a process cannot perform a τ-action), and $\varphi_3 = \texttt{max}\ X.([\tau][\alpha]\texttt{ff} \wedge [\tau]X)$ (*i.e.,* a process cannot perform an α-action after any non-empty sequence of τ-actions). Notice that $\varphi_3 \equiv [[\tau]][\alpha]\texttt{ff}$. Let L_1, L_2 be the LTSs described below, where $L_1 \leq_o L_2$:

$$L_1: \quad p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} p_2 \xrightarrow{\alpha} p_3 \qquad\qquad L_2: \quad p_0 \xrightarrow{\sigma} p_2 \xrightarrow{\alpha} p_3 \quad \text{and} \quad p_1 \xrightarrow{\sigma} p_2.$$

L_2 is a $\leq_o$-maximal obscuring of L_1: any LTS L' with $L_2 \leq_o L'$ will have to be exactly L_2 according to Properties 4.2 through 4.2. LTSs L_1 and L_2 are really instances of LTSs L and L' from Example 13, resp. Consider L_3 described below:

$$L_3: \quad p_0 \xrightarrow{\tau} p_2 \xrightarrow{\alpha} p_3 \quad \text{and} \quad p_1 \xrightarrow{\tau} p_2$$

where L_2 is also an obscuring of L_3, $L_3 \leq_o L_2$. We observe that φ_1 is not reliably monitorable according to Definition 14: $p_0 \in [\![\varphi_1]\!]_{L_1}$ and $p_0 \notin [\![\varphi_1]\!]_{L_3}$, so a monitor that reliably monitors for φ_1 would need to reject and not reject p_0 in L_2. On the other hand, both φ_2 and φ_3 are reliably monitorable *w.r.t.* $\leq_o$. Let

$$m_2 = \sigma.\texttt{no} + \tau.\texttt{no} \quad \text{and} \quad m_3 = \texttt{rec}\ x.(\sigma.\alpha.\texttt{no} + \sigma.x + \tau.\alpha.\texttt{no} + \tau.x)$$

Table 2 Instrumentation rules for myopic monitors.

$$\textsc{iMon}\frac{p \xrightarrow{\mu}_L p' \quad m \xrightarrow{\mu'}_M m' \quad \mu \leq \mu'}{m \triangleleft p \xrightarrow{\mu'}_{I(L,M)} m' \triangleleft p'} \qquad \textsc{iTer}\frac{p \xrightarrow{\mu}_L p' \quad \forall \mu' \geq \mu.\ m \not\xrightarrow{\mu'}_M}{m \triangleleft p \xrightarrow{\mu}_{I(L,M)} \mathtt{end} \triangleleft p'}$$

$$\textsc{iTran}\frac{m \triangleleft p \xrightarrow{\sigma}_{I(L,M)} m' \triangleleft p' \quad p' \xrightarrow{\mu}_L p'' \quad \mu \leq \sigma}{m \triangleleft p \xrightarrow{\sigma}_{I(L,M)} m' \triangleleft p''}$$

be monitors from the full monitoring system (M^δ, I^δ). According to Properties 4.2, 4.2, and 4.2, for all LTSs $L \leq_o L'$, where L is an LTS on $(\textsc{Act}, \{\tau\})$, and every process p in L we have that $p \xrightarrow{\tau}_L$ if and only if $p \xrightarrow{\tau}_{L'}$ or $p \xrightarrow{\sigma}_{L'}$; therefore, m_2 monitors for φ_2 on L from L'. A process p of L violates φ_3 iff $p(\xrightarrow{\tau}_L)^+ q \xrightarrow{\alpha}_L$ for some q; by Properties 4.2, 4.2, 4.2, $p(\xrightarrow{\tau}_L)^+ q \xrightarrow{\alpha}_L$ iff $p(\xrightarrow{\tau}_{L'} \cup \xrightarrow{\sigma}_{L'})^+ q \xrightarrow{\alpha}_{L'}$, and therefore, m_3 monitors for φ_3 on L from L'.

We first introduce myopic monitors, that are equivalent to full monitors on total obscurings.

▶ **Definition 20.** A *myopic monitor* on $(\textsc{Act}, \textsc{Sil})$ is defined by the grammar:

$$m, n \in \textsc{Mon}_\sigma ::= \quad \mathtt{end} \quad | \ \mathtt{no} \quad | \ \alpha.m \quad | \ \sigma.m \quad | \ m + n \quad | \ \mathtt{rec}\ x.m \quad | \ x.$$

A myopic monitor's LTS semantics is defined by the transition rules in Table 1; the resulting monitor LTS is $M^\sigma = \langle \textsc{Mon}_\sigma, (\textsc{Act}, \{\sigma\}), \rightarrow \rangle$. The instrumentation I^σ of myopic monitors is then defined by the rules in Table 2.

Rules iMon and iTer are similar to those for I^δ. The difference is that when the monitor is expecting a more obscure action (*i.e.,* σ), the instrumentation can pass along a possibly less obscure process action. So, the instrumentation may interpret τ-transitions as σ-transitions. Rule iTran is new, but the intuition behind it is similar: the instrumentation may interpret a (possibly mixed) sequence of τ- and σ-transitions as a single σ-transition, if that is what the monitor was expecting. On total obscurings, myopic monitors behave like full monitors.

▶ **Lemma 21.** *If L is a total obscuring on $(\textsc{Act}, \{\tau, \sigma\})$, then for every $m \in \textsc{Mon}_\sigma$ and process p of L,* $\mathbf{rej}_{\langle M^\sigma, I^\sigma, L\rangle}(m, p)$ *iff* $\mathbf{rej}_{\langle M^\delta, I^\delta, L\rangle}(m, p)$.

As we see in the following, we can further restrict the syntax of myopic monitors while preserving monitorability with respect to reliably monitorable formulas. The *σ-alternating myopic monitors* are the myopic monitors restricted to the following syntax:

$$m, n \in \textsc{Mon}_{alt} ::= \quad \mathtt{end} \quad | \ \mathtt{no} \quad | \ \sigma.\mathtt{no} \quad | \ \alpha.m \quad | \ \sigma.\alpha.m \quad | \ m{+}n \quad | \ \mathtt{rec}\ x.m \quad | \ x.$$

The resulting monitor LTS is called M^{alt} and it is a fragment of M^δ.

▶ **Corollary 22.** *If φ is a reliably monitorable formula on $(\textsc{Act}, \{\tau\})$, then there is a σ-alternating myopic monitor that monitors for φ on every LTS L on $(\textsc{Act}, \{\tau\})$ from every total obscuring of L.*

▶ **Example 23.** We revisit the LTSs $L_1 \leq_o L_2$ from Example 19. Let $m_4 = \sigma.\sigma.\alpha.\mathtt{no}$ be a myopic monitor but *not* a σ-alternating one. We see that m_4 rejects process p_0 in L_1, but not in L_2 since m_4 flags processes that perform *at least two* silent actions before performing α; this is *not* the case for p_0 in L_2. The constraint of σ-alternation ensures that the monitors are not allowed to count silent actions and thus rely on information that may be hidden in

a further obscuring of the LTS: the information that a monitor of the form $\sigma.\mathtt{no}$ or $\sigma.\alpha.m$ can analyse is "at least one" silent transition (and then α in the second case), which is information guaranteed to be preserved by Properties 4.2 and 4.2.

The following results describe how monitor rejections are preserved by the obscuring preorder.

▶ **Lemma 24.** *For each $m \in \text{Mon}_\sigma$ and each process p of L where $L \leq_o L'$, $\mathbf{rej}_{\langle M^\sigma, I^\sigma, L' \rangle}(m, p)$ implies $\mathbf{rej}_{\langle M^\sigma, I^\sigma, L \rangle}(m, p)$.*

▶ **Lemma 25.** *For every σ-alternating myopic monitor m and p a process of L where $L \leq_o L'$, $\mathbf{rej}_{\langle M^\sigma, I^\sigma, L \rangle}(m, p)$ implies $\mathbf{rej}_{\langle M^\sigma, I^\sigma, L' \rangle}(m, p)$.*

▶ **Corollary 26.** *If a σ-alternating myopic monitor monitors for a formula φ on LTS L on $(\text{Act}, \{\tau\})$ from an obscuring of L, then the monitor reliably monitors for φ on L.*

Thus, monitorability implies reliable monitorability for such monitors. We can now identify a maximal reliably monitorable fragment of μHML on $(\text{Act}, \{\tau\})$, which we call RsHML:

$$\theta, \chi \in \text{RsHML} ::= \quad \mathtt{tt} \quad | \, \mathtt{ff} \quad | \, [\tau]\mathtt{ff} \quad | \, [\alpha]\theta \quad | \, [[\tau]][\alpha]\theta \quad | \, \theta \wedge \chi \quad | \, \mathtt{max}\ X.\theta \quad | \, X.$$

▶ **Definition 27 (Reliable Monitor Synthesis).** We define a reliable monitor synthesis function $(\!|\cdot|\!)_r$ from RsHML to σ-alternating myopic monitors.

$$(\!|\mathtt{tt}|\!)_r = \mathtt{end} \qquad (\!|\mathtt{ff}|\!)_r = \mathtt{no} \qquad (\!|X|\!)_r = x \qquad (\!|[\tau]\mathtt{ff}|\!)_r = \sigma.\mathtt{no}$$

$$(\!|\psi_1 \wedge \psi_2|\!)_r = \begin{cases} (\!|\psi_1|\!)_r & \text{if } (\!|\psi_2|\!)_r = \mathtt{end} \\ (\!|\psi_2|\!)_r & \text{if } (\!|\psi_1|\!)_r = \mathtt{end} \\ (\!|\psi_1|\!)_r + (\!|\psi_2|\!)_r & \text{otherwise} \end{cases} \qquad (\!|[\alpha]\psi|\!)_r = \begin{cases} \mathtt{end} & \text{if } (\!|\psi|\!)_r = \mathtt{end} \\ \alpha.(\!|\psi|\!)_r & \text{otherwise} \end{cases}$$

$$(\!|\mathtt{max}\ X.\psi|\!)_r = \begin{cases} \mathtt{end} & \text{if } (\!|\psi|\!)_r = \mathtt{end} \\ \mathtt{rec}\ x.(\!|\psi|\!)_r & \text{otherwise} \end{cases} \qquad (\!|[[\tau]][\alpha]\psi|\!)_r = \begin{cases} \mathtt{end} & \text{if } (\!|\psi|\!)_r = \mathtt{end} \\ \sigma.\alpha.(\!|\psi|\!)_r & \text{otherwise} \end{cases}$$

▶ **Lemma 28.** *For every formula $\varphi \in \text{RsHML}$, $(\!|\varphi|\!)_r$ reliably monitors for φ.*

▶ **Definition 29 (Reliable Formula Synthesis).** We define a reliable formula synthesis function $\|\cdot\|_r$ from σ-alternating myopic monitors to RsHML.

$$\|\mathtt{end}\|_r = \mathtt{tt} \qquad \|\mathtt{no}\|_r = \mathtt{ff} \qquad \|x\|_r = X$$
$$\|\sigma.\mathtt{no}\|_r = [\tau]\mathtt{ff} \qquad \|\sigma.\alpha.m\|_r = [[\tau]][\alpha]\|m\|_r \qquad \|\alpha.m\|_r = [\alpha]\|m\|_r$$
$$\|m + n\|_r = \|m\|_r \wedge \|n\|_r \qquad \|\mathtt{rec}\ x.m\|_r = \mathtt{max}\ X.\|m\|_r$$

▶ **Lemma 30.** *For every σ-alternating myopic monitor m on $(\text{Act}, \{\tau, \sigma\})$, m reliably monitors for $\|m\|_r$.*

Theorem 31 presents the main result of this section. The first part follows from Lemmata 28 and 30 whereas the second part is a consequence of Lemma 30 and Corollaries 22 and 26.

▶ **Theorem 31.** *The monitoring system (M^{alt}, I^σ) on $(\text{Act}, \{\tau, \sigma\})$ reliably monitors for* RsHML *on $(\text{Act}, \{\tau\})$. Moreover,* RsHML *is the largest reliably monitorable fragment of μHML up to logical equivalence.*

We note that RsHML is also a fragment of ssHML, the maximally monitorable fragment of μHML identified in Section 3. Theorem 31 holds for every preorder $\leq_o$ that has the properties listed in Subsection 4.2. Therefore, it also holds for $\leq_c$ from Definition 16.

▶ **Example 32.** We return to the server from Examples 12 and 17. Notice that formula ψ is a RsHML-formula, and therefore it is reliably monitorable.

We conclude by noting that myopic monitors can also be described as a fragment of full monitors by replacing $\sigma.\mathtt{no}$ by $\tau.\mathtt{no}+\sigma.no$ and $\sigma.\alpha.m$ with $\mathtt{rec}\ x.(\tau.x+\sigma.x+\tau.\alpha.m+\sigma.\alpha.m)$ in any myopic monitor. However, myopic monitors provide a cleaner, more efficient description of the same monitors.

6 Conclusions

We developed a general framework for *reliable monitorability* for monitoring setups that obscure information about the internal behaviour of systems. The framework is described through a family of LTS preorders that satisfy natural properties (4.2 through 4.2 of Subsection 4.2). Via further assumptions (properties 4.2, 4.2, 4.2) that guarantee a certain level of information obscuring, we identified RsHML, a maximal *reliably* monitorable fragment of μHML. Then, we provided a monitoring system, (M^{alt}, I^{σ}) that reliably monitors for RsHML.

Related work. In [9], Dwyer *et al.* use the approach of combining several properties to be monitored to produce a composite property and then project this composite property onto a smaller set of observable actions. This sampling technique effectively "silences" some of the observable actions and focuses on the rest to reduce overhead without risking unsound monitoring. Their approach highlights the importance of silent actions for RV and the need for a framework to handle imperfect information about silent events.

In [5] Basin *et al.* consider the problem of monitoring over defective traces (called incomplete/disagreeing logs). They propose an augmented LTL specification language that permits reasoning about incompleteness and handling of inconsistencies. In some sense, this is related to our reliable monitors that are able to provide correct verdicts in the presence of event obscuring; however, the authors in [5] do not tackle issues related to monitorability.

In [21], Shi *et al.* consider the problem of monitoring a wireless network via a wireless sniffer. A wireless sniffer may introduce uncertainty over a monitoring setup, as the trace it detects may not necessarily be the actual trace of the system, due to the intrinsic unreliability of the wireless network. The authors in [21] thus develop a monitoring framework to tolerate such errors. In separate work [6], Basin *et al.* tackle the problem of distributed monitoring over a network which may produce delays and/or failures; they use a monitoring system based on a real-time three-valued logic that can track when an event took place. Their monitors may then need to draw appropriate conclusions under incomplete or scrambled information. Although we do not consider aspects such as event reordering, our work could serve as a basis for a better understanding of the level of obscuring these systems can tolerate.

Variations. Our system can be adjusted to describe diverse situations, by either weakening or strengthening the power of obscuring preorders. For instance, one can relax properties 4.2 to 4.2 to describe situations where there is a guarantee that the system reveals its internal behaviour at least partly and to a certain degree. For example, we can take $\leq_o$ to be the identity relation on LTSs, as it satisfies properties 4.2 to 4.2 and therefore is an obscuring preorder; then, the reliably monitorable properties would be all of ssHML and the corresponding monitoring system would be that of full monitors. A preorder between $\leq_c$ and $=$ would perhaps be more interesting.

References

1. Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *arXiv preprint arXiv:1611.10212*, 2016.
2. Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. On the complexity of determinizing monitors. In Arnaud Carayol and Cyril Nicaud, editors, *Implementation and Application of Automata - 22nd International Conference, CIAA 2017, Marne-la-Vallée, France, June 27-30, 2017, Proceedings*, volume 10329 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2017. doi:10.1007/978-3-319-60134-2_1.
3. Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007. doi:10.1017/cbo9780511814105.
4. S. Arun-Kumar and Matthew Hennessy. An efficiency preorder for processes. *Acta Inf.*, 29(8):737–760, 1992. doi:10.1007/BF01191894.
5. David A. Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zalinescu. Monitoring compliance policies over incomplete and disagreeing logs. In Shaz Qadeer and Serdar Tasiran, editors, *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, volume 7687 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2012. doi:10.1007/978-3-642-35632-2_17.
6. David A. Basin, Felix Klaedtke, and Eugen Zalinescu. Failure-aware runtime verification of distributed systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 590–603. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.590.
7. Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *Theor. Comput. Sci.*, 669:33–58, 2017. doi:10.1016/j.tcs.2017.02.009.
8. Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
9. Matthew B. Dwyer, Madeline Diep, and Sebastian G. Elbaum. Reducing the cost of path property monitoring through sampling. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*, pages 228–237. IEEE Computer Society, 2008. doi:10.1109/ASE.2008.33.
10. Adrian Francalanza. A theory of monitors - (extended abstract). In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2016. doi:10.1007/978-3-662-49630-5_9.
11. Adrian Francalanza. Consistently-detecting monitors. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.8.
12. Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceed-*

ings, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017. doi:10.1007/978-3-319-67531-2_2.

13 Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. On verifying hennessy-milner logic with recursion at runtime. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2015. doi:10.1007/978-3-319-23820-3_5.

14 Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the hennessy-milner logic with recursion. *Formal Methods in System Design*, 51(1):87–116, 2017. doi:10.1007/s10703-017-0273-z.

15 Adrian Francalanza and Aldrin Seychell. Synthesising correct concurrent runtime monitors. *Formal Methods in System Design*, 46(3):226–261, 2015. doi:10.1007/s10703-014-0217-9.

16 Matthew Hennessy. *Algebraic Theory of Processes*. Foundations of Computing. MIT Press, 1988.

17 Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

18 Kim Guldstrand Larsen. Proof systems for satisfiability in hennessy-milner logic with recursion. *Theor. Comput. Sci.*, 72(2&3):265–288, 1990. doi:10.1016/0304-3975(90)90038-J.

19 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009. doi:10.1016/j.jlap.2008.08.004.

20 Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

21 Jinghao Shi, Shuvendu K. Lahiri, Ranveer Chandra, and Geoffrey Challen. Wireless protocol validation under uncertainty. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2016. doi:10.1007/978-3-319-46982-9_22.

22 Yoriyuki Yamagata, Cyrille Artho, Masami Hagiya, Jun Inoue, Lei Ma, Yoshinori Tanabe, and Mitsuharu Yamamoto. Runtime monitoring for concurrent systems. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *Lecture Notes in Computer Science*, pages 386–403. Springer, 2016. doi:10.1007/978-3-319-46982-9_24.

Maintaining Reeb Graphs of Triangulated 2-Manifolds*†

Pankaj K. Agarwal[1], Kyle Fox[2], and Abhinandan Nath[3]

1 **Duke University, Durham, NC, USA**
pankaj@cs.duke.edu
2 **Duke University, Durham, NC, USA**
kylefox@cs.duke.edu
3 **Duke University, Durham, NC, USA**
abhinath@cs.duke.edu

Abstract

Let $\mathbb{M}$ be a triangulated, orientable 2-manifold of genus g without boundary, and let h be a height function over $\mathbb{M}$ that is linear within each triangle. We present a kinetic data structure (KDS) for maintaining the Reeb graph $\mathcal{R}$ of h as the heights of $\mathbb{M}$'s vertices vary continuously with time. Assuming the heights of two vertices of $\mathbb{M}$ become equal only $O(1)$ times, the KDS processes $O((\kappa + g)n \operatorname{polylog} n)$ events; n is the number of vertices in $\mathbb{M}$, and κ is the number of *external* events which change the combinatorial structure of $\mathcal{R}$. Each event is processed in $O(\log^2 n)$ time, and the total size of our KDS is $O(gn)$. The KDS can be extended to maintain an augmented Reeb graph as well.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Reeb graphs, 2-manifolds, topological graph theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.8

1 Introduction

Let $h : \mathbb{X} \to \mathbb{R}$ be a function over a manifold $\mathbb{X}$. We refer to h as the height function for simplicity. Given a height value ℓ, the *ℓ-level set* of h is the set of all points on $\mathbb{X}$ whose heights equal ℓ. A *contour* is a connected component of a level set. The *Reeb graph* [15] of h encodes the topological changes to the contours of ℓ-level sets as one continuously varies the value ℓ. Reeb graphs are the generalization of contour trees, which are trees that encode the contours of functions defined on zero-genus manifolds, to manifolds of non-zero genus, and contain cycles if the genus of $\mathbb{X}$ is not zero.

The "height" functions may measure any manner of things including temperature, pressure, brightness of points on a shape, etc. Reeb graphs have a variety of applications in graphics, data visualization, and shape matching and retrieval; see [5,10,20] for some of them. However, many of the potential quantities used for height functions including those listed above may change over time. New heat measurements may be taken, or a user interacting with the lighting on a shape may wish to change the brightness of certain locations. In this work, we develop an efficient data structure, using the kinetic data structure framework [4], for

* Work on this paper is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by ARO grant W911NF-15-1-0408, and by grant 2012/229 from the U.S.-Israel Binational Science Foundation.

† See https://users.cs.duke.edu/~abhinath/dynamicReebGraphs.pdf for a full version of the paper.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 8; pp. 8:1–8:14

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

maintaining the Reeb graph as the height function varies continuously. Note that despite the height function changing continuously, the combinatorial structure of the Reeb graph changes only at certain discrete times, called *events*. Our goal is to characterize and detect these events, and to update the Reeb graph efficiently at these events.

Related work. Shinagawa and Kunii [16] describe an $O(n^2)$-time algorithm for constructing the Reeb graph of a piecewise linear height function over a 2-manifold with n triangles. Cole-McLaughlin *et al.* [6] improved the running time to $O(n \log n)$. Later, $O(n \log n)$-time algorithms were designed for the more general case of piecewise linear functions over simplicial complexes with n vertices, edges, and triangles [12, 13]. Pascucci *et al.* [14] give an online algorithm for computing Reeb graphs. Doraiswamy and Natarajan [8] describe an algorithm that splits the input domain containing s saddles into regions whose Reeb graphs are loop-free, computes these Reeb graphs, then combines them to get the final Reeb graph in $O(n \log n + ns)$-time.

Most work on time-varying height functions has focused on contour trees. Sohn and Bajaj [18] and Szymczak [19] compute a mapping between the contour trees of a changing height function at successive discrete time steps, but they ignore combinatorial changes in the contour tree between the time steps and compute each new tree using a static algorithm. Agarwal *et al.* [2] provide a detailed characterization of the changes in the contour tree of a time-varying function over a terrain, and they describe a kinetic data structure (KDS) [4] that can update the contour tree in $O(\log n)$ time per major data structure modifying event. See [11] for a survey on other kinetic data structures.

Edelsbrunner *et al.* [9] actually characterize the combinatorial changes in the *Reeb graph* of a time-varying function; however, their approach only considers smooth functions on a manifold that is compactified to what is effectively a 3-sphere (hence the Reeb graph has no loops). Moreover, they assume that the height function over all time values is given in advance, and the algorithm takes $O(n)$ time to update the graph at each event.

Our work. *et al.* [2] to Reeb graphs of more general domains than the terrain. However, their results rely heavily on the planarity of terrains and the simple connectivity of their contour trees. Therefore, we consider domains that nearly feature both of these properties while still capturing far more general settings. Namely, we focus on piecewise linear height functions over triangulated 2-manifolds of low genus. For simplicity, we will assume these manifolds are orientable and lack boundary.

We describe the first KDS to efficiently maintain the Reeb graphs of such functions. Fix a triangulated 2-manifold $\mathbb{M}$ of genus g and let $h : \mathbb{M} \to \mathbb{R}$ be a height function. We assume the height of each vertex is a constant degree polynomial function of time and that the roots of these polynomials can be computed in $O(1)$ time[1]. Let κ be the number of combinatorial changes that occur to the Reeb graph of h. Our KDS processes $O((\kappa + g)\lambda_{O(1)}(n) \log n)$ events in the worst case, where $\lambda_{O(1)}(.)$ is the maximum length of a Davenport-Schinzel sequence of order $O(1)$ and is almost linear [3]. Each event can be processed in $O(\log^2 n)$ time, and the total size of our KDS is $O(gn)$. Note that if we are allowed $O(n^2)$ space, it is not too difficult to maintain the Reeb graph in $O(\text{polylog}\, n)$ time per event, since we can explicitly store the $O(n)$ combinatorially different level sets, each of size $O(n)$, and easily maintain them dynamically.

[1] Our algorithm defines events as those time instances where the relative ordering of vertices according to their function values changes. We can also simulate arbitrarily changing the height of a vertex v from h_1 to h_2 through a sequence of vertex swaps involving v and the vertices with function values lying between h_1 and h_2.

To obtain our data structure, we extend and simplify the detailed characterization of the combinatorial changes in the contour tree described by Agarwal *et al.* [2] to the case of Reeb graphs over 2-manifolds. One advantage of our analysis is that we no longer distinguish between the inside and outside of contours; indeed, doing so is impossible unless we define an outer face with which to define containment. We also keep our list of distinct changes to the Reeb graph as generic as possible, helping us avoid a potentially lengthy case analysis for situations our data structure's algorithms can handle implicitly.

On the other hand, we need to process more events than the $O(\kappa + n)$ needed for contour trees [2]. Moreover, because Reeb graphs are more general than contour trees, and the changes in Reeb graphs at an event are more complex, the data structure itself is more involved. A major component of Agarwal *et al.*'s data structure is to locate a given vertex v's contour on the contour tree. To do so, they find a pair of nodes μ_1 and μ_2 on the contour tree such that there is a unique height-monotone path between them containing v, and then they perform a binary search along the path. Unfortunately, the presence of loops in a Reeb graph means that it may contain multiple such paths between a pair of nodes. To solve this problem, we explicitly maintain g level sets of the height function and a spanning tree of the Reeb graph. Using the level sets, we can find a pair μ_1 and μ_2 as before such that the unique height-monotone path *in the spanning tree* contains v. We take advantage of the topology of 2-manifolds to efficiently maintain these level sets. In particular, we recognize that contours essentially form cycles in the *dual graph* of a manifold, and as the level sets change these cycles should and can be modified easily.

The rest of the paper is organized as follows. We briefly discuss the necessary background in Section 2. We introduce our model of time-varying height functions, and give a characterization of the changes in the Reeb graph in Section 3. We describe a KDS, including a few useful operations for maintaining a dynamic Reeb graph in Section 4, and we describe how to efficiently update the KDS in Section 5.

2 Preliminaries

2-manifolds. A triangulated 2-manifold $\mathbb{M} = (V, E, F)$ without boundary consists of a set of triangles F bounded by a set of edges E incident to a set of vertices V. Let $n = |F|$. We require each edge to bound two distinct triangles and to contain two distinct incident vertices. Thus, every point on $\mathbb{M}$ has an open neighborhood homeomorphic to the plane $\mathbb{R}^2$. A 2-manifold is orientable if it does not contain a subset homeomorphic to the Möbius band. We consider only orientable 2-manifolds in this paper. Let $\chi = |V| - |E| + |F|$ denote the *Euler characteristic* of $\mathbb{M}$. The *genus* g of a 2-manifold is the maximum number of disjoint simple cycles through the manifold that can be removed without disconnecting it. By Euler's formula, we have $\chi = 2 - 2g$. For simplicity, we assume every vertex v has degree $O(1)$. Our data structure can be modified easily to handle arbitrary vertex degree; however the processing time of events will increase proportionally to the degree of vertices involved in the event. Let $\mathbb{M}^*$ denote the *dual graph* of $\mathbb{M}$. The graph $\mathbb{M}^*$ contains a vertex for every triangle of $\mathbb{M}$, and two of its vertices are adjacent if and only if their corresponding triangles share an edge in $\mathbb{M}$. Let $h : \mathbb{M} \to \mathbb{R}$ be a *height function*. We assume that the restriction of h to each triangle of $\mathbb{M}$ is linear and that the heights of all vertices are distinct.

Critical points, level sets, and contours. The *link* of v, denoted by $\mathrm{Lk}(v)$, is the cycle formed by those edges of $\mathbb{M}$ whose endpoints are neighbors of v. The lower/down (resp. higher/up) link of v, $\mathrm{Lk}^-(v)$ (resp. $\mathrm{Lk}^+(v)$), is the subgraph of $\mathrm{Lk}(v)$ induced by vertices

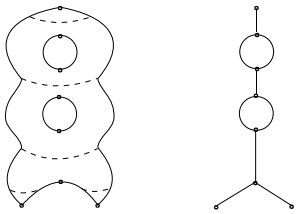


Figure 1 Reeb graph $\mathcal{R}$ on 2-manifold $\mathbb{M}$; the function h is given by height from the base.

u with $h(u) < h(v)$ (resp. $h(u) > h(v)$). The down (resp. up) edges of v are the edges between v and its lower (resp. upper) link. A *minimum* (resp. *maximum*) of $\mathbb{M}$ is a vertex v for which $\mathrm{Lk}^-(v)$ (resp. $\mathrm{Lk}^+(v)$) is empty. Such a vertex is also called an *extremal* vertex. A non-extremal vertex v is *regular* if $\mathrm{Lk}^-(v)$ (and also $\mathrm{Lk}^+(v)$) is connected, and *saddle* otherwise. A vertex that is not regular is called a *critical* vertex.

For $\ell \in \mathbb{R}$, the *ℓ-level set* of $\mathbb{M}$, denoted by $\mathbb{M}_\ell$, consists of points $x \in \mathbb{M}$ with $h(x) = \ell$. We refer to a level set $\mathbb{M}_\ell$ where $\ell = h(v)$ for some critical vertex v as a *critical level*. A *contour* of $\mathbb{M}$ is a connected component of a level set of $\mathbb{M}$. Each vertex $v \in V$ is contained in exactly one contour in $\mathbb{M}_{h(v)}$, which we call *the contour of v*. A contour not passing through a critical vertex is a simple polygonal cycle. A contour passing through an extremal vertex is a single point, and a contour passing through a saddle v consists of two or more simple cycles, one per component of $\mathrm{Lk}^-(v)$, with v being their only intersection point. A contour C not passing through a vertex can be represented by the cyclic sequence of edges of $\mathbb{M}$ that it passes through. Similarly, a contour C passing through a vertex can be represented by zero or more sequences of edges, depending on whether the vertex is critical, where each sequence represents a cycle of C.

Let $\varepsilon = \varepsilon(\mathbb{M})$ denote a positive value smaller than the height difference between any two vertices of $\mathbb{M}$. An *up-contour* (resp. *down-contour*) of a saddle vertex v is any contour of $\mathbb{M}_{h(v)+\varepsilon}$ (resp. $\mathbb{M}_{h(v)-\varepsilon}$) that intersects an edge incident on v. If v has two up-contours (resp. down-contours) and one down-contour (resp. up-contour) it is called a *positive (resp. negative) saddle vertex*. A saddle vertex v that is either negative or positive (but not both) is called a *simple* saddle (this also means that $\mathrm{Lk}^-(v)$ and $\mathrm{Lk}^+(v)$ each consist of two connected components). For simplicity, we assume that each saddle vertex v is *simple*. As before, our data structure can be modified easily to handle vertices that are not simple.

Reeb graphs. Consider raising ℓ from $-\infty$ to ∞. The contours continuously deform, but the level set's topology does not change while $\mathbb{M}_\ell$ varies between two consecutive critical levels. A new contour appears as a single point at a minimum vertex, and an existing contour contracts into a single point and disappears at a maximum vertex. An existing contour (the down-contour of v) splits into two new contours (the up-contours of v) at a positive saddle vertex v, and two contours (the down-contours of v) merge into one contour (the up-contour of v) at a negative saddle vertex v. The *Reeb graph* $\mathcal{R}$ of h is a graph on the critical vertices of $\mathbb{M}$ that encodes these topological changes of the level set. An arc (v, w) of $\mathcal{R}$ *represents* a contour that appears at v and disappears at w. If $\mathbb{M}$ has genus g, then $\mathcal{R}$ is a tree with exactly g additional arcs [6] ($\mathcal{R}$ may contain *parallel arcs* between two nodes; each such arc is considered distinct). We will refer to vertices and edges in $\mathbb{M}$ and *nodes* and *arcs* in $\mathcal{R}$.

More formally, two contours C_1 and C_2 at heights ℓ_1 and ℓ_2, respectively, are called *equivalent* if C_1 and C_2 belong to the same connected component of $\Gamma = \{x \in \mathbb{M} \mid \ell_1 \leq h(x) \leq \ell_2\}$ and that component of Γ does not contain any critical vertex. An equivalence

class of contours starts and ends at critical vertices. If a class starts at a critical vertex v and ends at w, then (v, w) is an arc in $\mathcal{R}$. We refer to v as a *down neighbor* of w and to w as an *up neighbor* of v. A down arc (resp. up arc) of v goes to another node on $\mathcal{R}$ of lesser (resp. greater) height. Equivalently, $\mathcal{R}$ is the quotient space in which each contour is represented by a point and connectivity is defined in terms of the quotient topology. Let $\rho : \mathbb{M} \to \mathcal{R}$ be the associated quotient map, which maps all points of a contour to a single point on $\mathcal{R}$. Fix a point p in $\mathbb{M}$. If p does not lie in the contour of a critical vertex, $\rho(p)$ lies in the relative interior of an arc in $\mathcal{R}$; if p is an extremal vertex, $\rho(p)$ is a leaf node of $\mathcal{R}$; and if p lies in the contour of a saddle vertex then $\rho(p)$ is a non-leaf node of $\mathcal{R}$. See Figure 1. We extend the height function h to $\mathcal{R}$ by defining $h(\rho(p)) = h(p)$ for all $p \in \mathbb{M}$.

Each node of $\mathcal{R}$ is labeled with the corresponding critical vertex of $\mathbb{M}$. A non-leaf node of $\mathcal{R}$ is called *positive* (resp. *negative*) if its corresponding saddle vertex is positive (resp. negative). The combinatorial description of $\mathcal{R}$ is the set of its nodes along with their labels, and the set of its arcs.

3 Time varying Reeb graph

We use the same model for the time varying function as the one in [2]. Specifically, we have a one-parameter family of height functions over $\mathbb{M}$, $h : \mathbb{M} \times \mathbb{R} \to \mathbb{R}$, where the extra dimension in the domain is time. We assume that at any given time at most two vertices have equal height value, and the heights of any two vertices become equal $O(1)$ times. Finally, if $h(u, t_0) = h(v, t_0)$ then the function $h(u, t) - h(v, t)$ changes sign at $t = t_0$. We now consider the different changes that may occur in $\mathcal{R}$ as time passes. We call each time t where the topology or node labels of $\mathcal{R}$ change an *external event*. We focus on external events in this section. In later sections, we define other types of events that only affect our data structure but not $\mathcal{R}$ itself; we refer to these events as *internal events*. If an event occurs at time t, we let t^- (resp. t^+) denote the time immediately before (resp. after) the event, i.e., $t^- = t - \delta$ (resp. $t^+ = t + \delta$) for some arbitrarily small $\delta > 0$.

We characterize external events by whether or not they change the set of nodes in $\mathcal{R}$. We refer to events that do change the nodes of $\mathcal{R}$ as *node* events. Recall that the nodes of $\mathcal{R}$ are the images of critical vertices of $\mathbb{M}$. Therefore, a node $\rho(u)$ enters or leaves $\mathcal{R}$ only when the uplinks or downlinks of u in $\mathbb{M}$ change their connectivity or when u becomes an extremum. These events occur only at a time t for which $h(u, t) = h(v, t)$ for some neighbor v of u. Immediately before and after the node event, no other vertex has the topology of its links changed, so only $\rho(u)$ and $\rho(v)$ can be added or removed from $\mathcal{R}$.

Changes to $\mathcal{R}$ that do not change the set of nodes only occur when two adjacent non-leaf nodes in $\mathcal{R}$ go to the same height value and the number of arcs remains the same, and are referred to as *interchange* events. We now explore both types of events in more detail.

Node events. Suppose a node $\rho(u)$ enters or leaves $\mathcal{R}$ at a time t, and let v be the neighbor of u for which $h(u, t) = h(v, t)$. Suppose u becomes a saddle vertex so that $\rho(u)$ is added to $\mathcal{R}$ with three incident arcs. One of two situations may occur. First, $\rho(u)$ subdivides an arc of $\mathcal{R}$, adding one additional arc to $\mathcal{R}$. In this case, $\rho(v)$ must enter $\mathcal{R}$ as well with only a single arc incidence. In other words, v is a new extremal vertex. We call this node event a *birth* event. Second, $\rho(u)$ replaces a node of $\mathcal{R}$ which must be $\rho(v)$. We call this type of node event a *shift* event (shift events occur because $\mathbb{M}$ is a piecewise-linear manifold; they do not occur for smooth surfaces). In opposition to the birth events, both $\rho(u)$ and $\rho(v)$ may leave $\mathcal{R}$. The removal of a saddle and extremal vertex is a *death* event. See Figure 2.

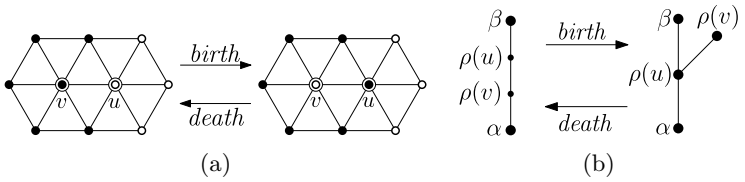


Figure 2 Birth/death event : (a) shows the changes in $\mathbb{M}$, and (b) shows the changes in $\mathcal{R}$. Solid vertices have lesser height than u and v; empty vertices have greater height. Figures are from [2].

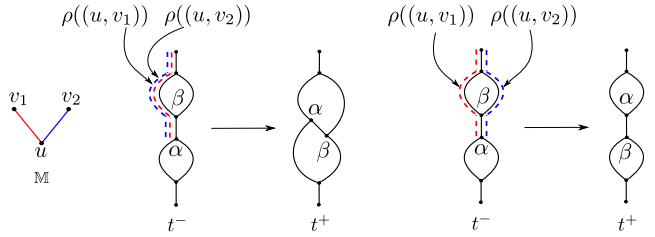


Figure 3 An interchange event where α is negative and β is positive at time t^-. We have $\alpha = \rho(u)$, and v_1 and v_2 are two vertices, one from each uplink of u. The figure shows the two possible ways in which (u, v_1) and (u, v_2) can map to $\mathcal{R}$, and how it determines the changes in $\mathcal{R}$.

Interchange events. The combinatorial changes that occur in $\mathcal{R}$ during node events are easy to deduce simply by looking at which vertices are involved in each event and examining the changes to their links' topology. However, interchange events are more subtle.

Suppose an interchange event occurs at time t where for an arc (α, β) we have $h(\alpha, t^-) < h(\beta, t^-)$ and $h(\beta, t^+) < h(\alpha, t^+)$. Only the neighborhood of α and β changes, because the relative ordering over vertices mapping to arcs outside the neighborhood do not change. We will focus on what the new up-arcs for α should be; other new up-arcs to neighbors of α and β must come from β. The case for the new down-arcs of β is symmetric. The following lemma describes the up-arcs based purely on which arcs of $\mathcal{R}$ incident to α and β intersect the projections of up-edges emanating from u. Note that in the presence of genus, there may be parallel arcs between a pair of nodes undergoing an interchange event. The two arcs are still present after the event, but their endpoints switch labels.

▶ **Lemma 1.** *Let (α, β) be an arc at time t^- such that $h(\alpha, t^-) < h(\beta, t^-)$ and $h(\beta, t^+) < h(\alpha, t^+)$. Let ξ be a node such that $h(\xi, t) > h(\alpha, t) = h(\beta, t)$. There exists one arc (α, ξ) at time t^+ for each instance of the following occurrences: either (i) there exists an arc (α, ξ) at time t^-, or (ii) there exists an arc (β, ξ) at time t^- such that there is an up-edge (u, v) of vertex u with $\alpha = \rho(u)$ and $\beta \neq \rho(v)$ such that $\{\beta\} \subsetneq \rho((u, v)) \cap (\beta, \xi)$.*

See Figure 3 for an example, where α is negative and β is positive at time t^-, and the two ways in which $\mathcal{R}$ can change. We remark that in condition *(ii)*, the requirement states that $\{\beta\}$ is a *proper* subset of $\rho((u, v)) \cap (\beta, \xi)$. See the full paper for the proof of Lemma 1.

According to Lemma 1, the changes to $\mathcal{R}$ during an interchange event between nodes $\alpha = \rho(u)$ and $\beta = \rho(u')$ can be completely determined after figuring out how the maps of edges incident to u and u' intersect arcs incident to α and β. In the next section, we present a kinetic data structure that aids in determining these intersections.

4 KDS for $\mathcal{R}$

We begin with a high level overview of our KDS for maintaining $\mathcal{R}$, including details on some auxiliary data structures we use. Our data structure, although similar in some aspects to the one of Agarwal *et al.* [2], is necessarily more complicated to handle the additional topology present in $\mathbb{M}$ and $\mathcal{R}$. Unlike in their KDS, ours explicitly maintains a collection of g level sets. These level sets are chosen so that there is at most one path between any pair of points in $\mathcal{R}$ that avoids crossing the level sets. We use this property in the following way: in order to compute changes to $\mathcal{R}$ during external events, we need to be able to locate $\rho(v)$ on $\mathcal{R}$ for any vertex v. This is accomplished by computing the first extremal vertices or the aforementioned level sets reached while greedily walking along up-edges (resp. down-edges) from v and then searching for the unique arc at v's height in the aforementioned path in $\mathcal{R}$ between these positions.

Our data structure maintains the Reeb graph $\mathcal{R}$, a spanning tree $\mathcal{T}$ of $\mathcal{R}$, and the set $\mathcal{L} = \mathcal{R} - \mathcal{T}$ of *loop arcs*. Set $\mathcal{L}$ contains exactly g arcs [6, Lemma A]. For each loop arc, we select a vertex $x \in \mathbb{M}$ such that $\rho(x)$ lies in the *closure* of the loop arc (i.e., the arc along with its two endpoints). We designate this set of vertices $\mathcal{V}$ as the *representative vertices* of $\mathcal{L}$. Since the maximum degree of $\mathcal{R}$ is at most 3, a single node may be incident to up to 2 loop arcs. That node's saddle vertex may therefore be representative for up to 2 loop arcs.

We store $\mathbb{M}$ in the form of a doubly connected edge list (DCEL) [7]. For each negative (positive) saddle vertex, we associate with each connected component of its down (up) link the arc of $\mathcal{R}$ holding down(up)-contours through the component. We maintain an *event queue* as a priority queue that, at any point in time, helps us to efficiently compute the next event to occur. Finally, we maintain two kinds of auxiliary data structures detailed below: (i) the *crossing level sets* which we define as the level sets at the same heights as representative vertices, and (ii) a forest of *descent and ascent trees*.

Crossing level sets. Let x be a representative vertex. We refer to the $h(x)$-level set $\mathbb{M}_{h(x)}$ as x's *crossing level set*. We explicitly maintain the edges intersected internally by x's crossing level set. For each regular contour avoiding x, we store a single *anchor edge* e and a path through $\mathbb{M}^*$ between e's incident triangles (which are vertices in $\mathbb{M}^*$) containing all triangles and edges other than e intersected by the contour (see Figure 4). For the contour passing through x, we store one or two paths (depending on whether x is a saddle) through $\mathbb{M}^*$ containing all the triangles and edges intersected internally by the one or two cycles in x's contour (see Figure 4). These paths are all stored in a single forest of edge and vertex-disjoint trees. Note that each edge and triangle of $\mathbb{M}$ is intersected by 0 to g crossing level sets. We store distinct copies of the edges and triangles in each crossing level set intersecting them internally. For each edge, we store a list of its copies ordered by height of their crossing level set. Let $\mathrm{Cr}_-(e)$ and $\mathrm{Cr}^-(e)$ denote the lowest and highest copies of e respectively. The total size of these crossing level sets is $O(gn)$.

We annotate crossing level set contours with the arcs/nodes they lie on in $\mathcal{R}$. If a crossing level set contour does not contain a vertex or passes through a regular vertex, then it is simply annotated with the arc in $\mathcal{R}$ containing its image under ρ. If the contour passes through a saddle vertex x, the annotations are slightly different. Suppose x is a negative (positive) saddle vertex. Let $C(x)$ be x's contour, and let C_1 and C_2 be x's two down(up)-contours, corresponding to the two down (up) arcs a_1 and a_2 of $\rho(x)$ respectively. Observe that each cycle of $C(x)$ shares intersected edges with exactly one of C_1 and C_2. For $i \in \{1, 2\}$, we annotate the contour cycle sharing edges with C_i with the arc a_i. Finally, we annotate each

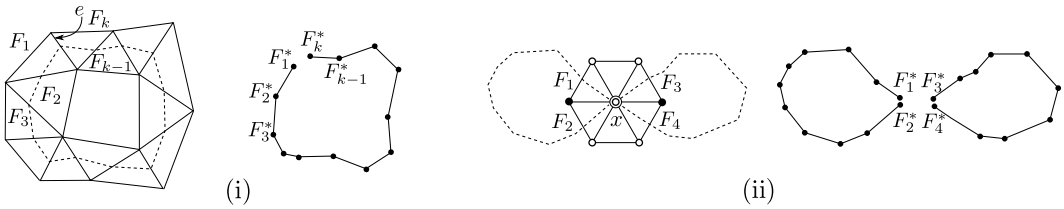


Figure 4 (i) (left) A crossing level set contour avoiding vertices of $\mathbb{M}$, along with its anchor edge e; (right) the contour is stored as a path in the dual graph between e's adjacent triangles. (ii) (left) A crossing level set contour passing through saddle vertex x; (right) the contour is stored as two paths. Recall that the dual of a triangle $F \in \mathbb{M}$ is a vertex $F^* \in \mathbb{M}^*$.

contour with the crossing level set it belongs to, and separately annotate each crossing level set with its representative vertex so it can be changed easily. Later, we describe how to use these annotations to take any edge e of $\mathbb{M}$, discover which crossing level sets intersect e, and learn which arc(s) of $\mathcal{R}$ contain the intersection and its neighborhood.

Finally, for each representative vertex x, we store two *dynamic and kinetic tournaments* $\psi^-(x)$ and $\psi^+(x)$ for vertices incident to edges crossed by x's crossing level set [1]. Tournament $\psi^-(x)$ stores such vertices u with $h(u) < h(x)$. At any point in time, $\psi^-(x)$ can be efficiently queried to provide the highest vertex lower than x. We call this vertex the *winner* of $\psi^-(x)$. The other tournament $\psi^+(x)$ stores the vertices u with $h(u) > h(x)$ and can be queried for its winner, i.e., the lowest vertex in $\psi^+(x)$ higher than x. Vertices can be added or removed from $\psi^-(x)$ and $\psi^+(x)$ as required by our data structure. Maintaining the tournaments introduces additional internal *tournament events*. We discuss the running times associated with modifying and maintaining the tournaments later.

Recall that a single vertex x may be the representative of up to two loop arcs. In this case, we store two copies of x's crossing level set so that they can be modified independently as our data structure requires.

Descent and ascent trees. We maintain a forest of rooted descent and ascent trees [2]. The descent trees are defined as follows. Each vertex of $\mathbb{M}$ appears exactly once in the set of descent trees. We also include a collection of special root vertices, called *level set roots*, that each have exactly one child and a label to an edge of $\mathbb{M}$. For each vertex $u \in \mathbb{M}$, if u is not a minimum and not a representative vertex, then we choose an arbitrary vertex $w \in \mathrm{Lk}^-(u)$. If edge (u, w) does not intersect a crossing level set internally, then w is made u's parent in the descent tree. Otherwise, we create a level set root c, set c to be u's parent, and record on c the edge (u, w) that would have been added to the descent trees if the level set were not crossed. Ascent trees are constructed in a similar manner by taking arbitrary vertices from uplinks and either adding edges to them or adding edges to level set roots. Note that the roots of our ascent and descent trees can be extremal vertices, representative vertices, or level set roots.

Final details. The spanning tree $\mathcal{T}$ of $\mathcal{R}$, the paths representing contours of the crossing level sets, and the ascent and descent trees are all stored as (rooted) *link-cut trees*. Link-cut trees support many basic tree operations in $O(\log n)$ time each [1, 17]. See the full paper

[2] Our definitions of descent and ascent trees differ from those of Agarwal *et al.* [2] due to the existence of crossing level sets.

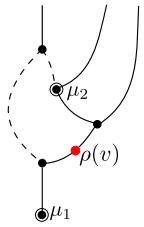


Figure 5 A height monotone path between μ_1 and μ_2 through $\mathcal{T}$ and containing $\rho(v)$. The dotted arcs are loop arcs.

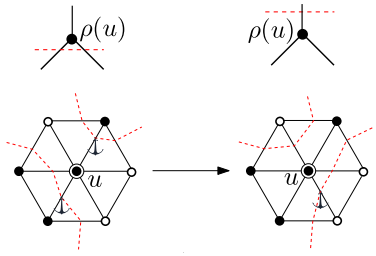


Figure 6 An LSCross(x, u) operation where u is a negativesaddle. The red dotted line represents x's crossing level set, and the anchor edges are shown.

for further details. It is possible to look up or store a constant amount of information on individual trees of a link-cut tree forest (such as a contour's anchor edge) in $O(\log n)$ time per operation.

Initialization. The Reeb graph $\mathcal{R}$, spanning tree $\mathcal{T}$, loop arcs $\mathcal{L}$, and the set $\mathcal{V}$ can be computed in $O(n \log n)$ time [6]. The ascent and descent trees can be initialized in $O(n \log n)$ time. The edges in the crossing level set of a vertex $x \in \mathcal{V}$ can be computed by traversing the edges of $\mathbb{M}$ crossing height $h(x)$ in $O(n)$ time, and can be stored as link-cut trees and tournaments $\psi^-(x)$ and $\psi^+(x)$ in $O(n \log n)$ time [1, 17]. There are at most g such level sets, so overall the time taken is $O(gn \log n)$.

4.1 Useful operations

We define the following operations used by our data structure. The first two are useful in determining the combinatorial changes to $\mathcal{R}$ during external events. The latter two aid us in maintaining crossing level sets as their heights change.

- FindArc(v) : Returns the arc of $\mathcal{R}$ containing $\rho(v)$ for a regular vertex v.
- LocateArc(α, β, v) : Suppose $(\alpha, \beta) \in \mathcal{R}$ is an arc and no other vertex has height between $h(\alpha)$ and $h(\beta)$. Let (u, v) be an edge of $\mathbb{M}$ with $\alpha = \rho(u)$, $\beta \neq \rho(v)$, and $h(\beta)$ lies between $h(\alpha)$ and $h(v)$. The operation LocateArc(α, β, v) returns the only arc $(\beta, \beta') \in \mathcal{R}$ incident to β, other than (α, β), such that $\{\beta\} \subsetneq (\beta, \beta') \bigcap \rho(u, v)$.
- LSCross(x, u): Let $x \in \mathcal{V}$ and u be any other vertex such that $h(x, t) = h(u, t)$. The operation LSCross(x, u) updates x's crossing level set(s) from intersecting down edges (up edges) of u at time t^- to intersecting up edges (down edges) of u at time t^+.
- Handoff(x, x'): Let $x \in \mathcal{V}$ and x' be any other non-extremal vertex such that no other vertex has height between $h(x)$ and $h(x')$. The operation Handoff(x, x') updates one crossing level set of x from containing x and intersecting the down edges (up edges) of x' to containing x' and intersecting the up edges (down edges) of x. Vertex x' is then made a representative owning the crossing level set.

We now sketch how to implement each operation. The details are given in the full paper.

FindArc(v) and LocateArc(α, β, v): For FindArc(v), we want to find a pair of nodes μ_1 and μ_2 bounding a height monotone path through $\mathcal{T}$ that contains $\rho(v)$. Afterward, we can binary search for the projection of v on that path. See Figure 5 for an example. The high level idea behind our strategy to find μ_1 is to search for the root of v's descent tree, and use it to find an extremal vertex reachable via a height monotone path through $\mathcal{T}$ or to find

an arc of $\mathcal{R}$ reachable in a similar way; the higher of the two nodes on the arc is reachable through $\mathcal{T}$. We compute μ_2 symmetrically. For LocateArc(α, β, v), we use a procedure similar to that of FindArc to quickly find a height monotone path in $\mathcal{T}$ from α that passes through β and $\rho(v)$.

▶ **Lemma 2.** *Both* FindArc(v) *and* LocateArc(α, β, v) *can be implemented in* $O(\log n)$ *time.*

LSCross(x, u) and Handoff(x, x'): For LSCross(x, u), our goal is to remove down edges of u from x's crossing level set one-by-one (assuming $h(x, t^-) < h(u, t^-)$ without loss of generality) using link-cut tree operations before adding the up edges of u to x's crossing level set. The details on which edges to link, unlink, or designate as anchors vary depending on whether u is an extremal vertex, a saddle, or a regular vertex. See Figure 6. Handoff(x,x') is also implemented in a similar fashion.

▶ **Lemma 3.** *Both* LSCross(x, u) *and* Handoff(x, x') *can be implemented in* $O(\log^2 n)$ *time.*

5 Handling events

Internal events and certificates. We define a few internal events that are needed to keep its auxiliary structures accurate. A *local event* occurs whenever two adjacent vertices in $\mathbb{M}$ have equal heights. A *crossing event* occurs whenever a representative vertex x and any vertex $u \neq x$ have equal height. A *winner event* occurs when the winner of a representative's tournament changes. Finally, a *tournament event* occurs when any of the representative vertices' tournaments need to be repaired as described by Agarwal *et al.* [1].

We maintain a collection of certificates that certify our data structure to be accurate. When a certificate fails, an event may occur. We maintain certificates of the following types:

- For each edge of $\mathbb{M}$, a certificate that fails when its two endpoints' heights become equal.
- For every arc of $\mathcal{R}$, a certificate that fails when its two endpoints' heights become equal.
- For every representative vertex $x \in \mathcal{V}$ and its two tournaments $\psi^-(x)$ and $\psi^+(x)$, a certificate that fails when the heights of x and the tournament winner become equal. One can easily verify that one of these two certificates for x will fail any time the height of any vertex $u \neq x$ becomes equal to $h(x)$.
- For every representative vertex $x \in \mathcal{V}$, a collection of certificates as described by Agarwal *et al.* [1] certifying the correctness of the tournaments $\psi^-(x)$ and $\psi^+(x)$.

Note that multiple certificate failures can occur at the same time, since the same two vertices can be involved in the failure of multiple certificates. For instance, during a death event, the certificates for both an arc of $\mathcal{R}$ and an edge of $\mathbb{M}$ will fail.

Repairing the KDS. It is straightforward from the definition of the different events to know what type of event is occurring with each certificate failure. We now discuss how our data structure handles events. Each event involves only a constant number of vertices (under our assumption that the vertex degree is constant).

To avoid opening the 'black box' we leave the handling of tournament events and the repairing of tournaments to the procedures described by Agarwal *et al.* [1]. Winner events are handled separately and in addition to the other type of events in $O(\log n)$ time by simply updating the certificates for representative vertices and their tournaments with new winners.

We now describe how to handle the remaining types of events, beginning with external events, and then focusing on non-external crossing events. We conclude with updating the descent and ascent trees, a process that must be done after every event. In general, we handle events by performing local changes that guarantee $\mathcal{T}$ is still a spanning tree of $\mathcal{R}$. We must be sure to appropriately hand off crossing level sets so their representative vertices still lie on loop arcs in $\mathcal{L}$. In addition, we must make sure that each edge still has its crossing level set copies ordered by height. To do so, we will generally perform HANDOFF operations so that only one vertex involved in each event represents crossing level sets. LSCROSS operations can then be performed safely (as we essentially move from time t^- to time t^+) before doing some number of HANDOFF operations to guarantee each vertex again represents the correct number of crossing level sets.

Shift event: Suppose u is critical and v is regular at t^-, and vice versa at t^+. We relabel node $\rho(u)$ in $\mathcal{R}$ by v. Node $\rho(v)$ is now incident to every loop arc that originally contained $\rho(u)$, so we call HANDOFF(u, v) for each crossing level set that belongs to u (recall, a vertex can be a representative up to two times). Finally, we call LSCROSS(v, u) if v is now a representative, because its (newly acquired) crossing level sets pass over u at time t.

Birth event: Suppose u and v are not critical at t^-, but they become critical at t^+. Let v and u be a maximum and positive saddle respectively at time t^+ (the other cases are similar). We need to figure out where the new arc $(\rho(u), \rho(v))$ will lie in $\mathcal{R}$ and possibly hand off a crossing level set from v, because $\rho(v)$ will soon be a leaf of $\mathcal{R}$. We call FINDARC(u) to figure out which arc (ξ, ζ) with $h(\xi) < h(\zeta)$ contains u and v. If $(\xi, \zeta) \in \mathcal{T}$, then we remove (ξ, ζ) from $\mathcal{T}$, and add arcs $(\xi, \rho(u))$, $(\rho(u), \zeta)$, and $(\rho(u), \rho(v))$ to $\mathcal{T}$ using link-cut tree operations. We are done, because $u, v \notin \mathcal{V}$ in this case.

Suppose $(\xi, \zeta) \in \mathcal{L}$. If $v \in \mathcal{V}$, we call HANDOFF(v, u). If u now has a crossing level set, we call LSCROSS(u, v). We replace (ξ, ζ) in $\mathcal{L}$ by $(\rho(u), \zeta)$, and add $(\xi, \rho(u)), (\rho(u), \rho(v))$ to $\mathcal{T}$. For the case of $u, v \notin \mathcal{V}$, we must guarantee the representative vertex w for (ξ, ζ) still lies on a loop arc. If $h(w) > h(u)$ (resp. $h(w) < h(v)$) at t^-, replace (ξ, ζ) in $\mathcal{L}$ by $(\rho(u), \zeta)$ (resp. $(\xi, \rho(u))$), and add $(\rho(u), \rho(v))$ and $(\rho(u), \xi)$ (resp. $(\zeta, \rho(u))$) to $\mathcal{T}$.

Death event: Suppose at t^-, $\alpha = \rho(u)$ was a positive saddle and $\beta = \rho(v)$ was a maximum, and they both become regular at t^+ (the other cases are similar). We have $(\alpha, \beta) \notin \mathcal{L}$, and in particular, v is not a representative vertex. We remove arc (α, β) from $\mathcal{T}$. Let $(\xi, \alpha), (\alpha, \zeta)$ be the other two arcs adjacent to α with $h(\xi) < h(\zeta)$. We cannot have both (ξ, α) and (α, ζ) as loop arcs at t^-. If neither arc is a loop arc or $(\xi, \alpha) \in \mathcal{L}$ then we relabel (ξ, α) to be (ξ, ζ) and remove (α, ζ) from $\mathcal{T}$. Otherwise, we relabel (α, ζ) to be (ξ, ζ) and remove (ξ, α) from $\mathcal{T}$. If either arc was a loop arc, then the new loop arc still contains the image of its representative. Finally, if $u \in \mathcal{V}$, we call LSCROSS(u, v).

Interchange event: Consider an interchange event between two nodes $\alpha = \rho(u)$ and $\beta = \rho(v)$ where $h(\alpha, t^-) < h(\beta, t^-)$. We must first figure out the new incident arcs of α and β. Afterward, we will pick which incident arcs will be designated as loop arcs and finally move crossing levels as appropriate to guarantee the new loop arcs have their representatives. We pick two up-neighbors w_1, w_2 of u such that $\rho(w_i) \neq \beta$ for $i \in \{1, 2\}$, one from each uplink of u. We then figure out the new up arcs of α using LOCATEARC(α, β, w_i) for $i \in \{1, 2\}$ (see Lemma 1). Similarly, we figure out the new down arcs of β, thus giving us the combinatorial changes in $\mathcal{R}$.

We define the *outer nodes* as the nodes incident to α and β at time t^- other than α and β themselves. Suppose α and β share a single arc in $\mathcal{R}$. Further suppose $(\alpha, \beta) \notin \mathcal{L}$ at t^-. Thus, $(\alpha, \beta) \in \mathcal{T}$. If we contract (α, β), $\mathcal{R}$ is identical at t^- and t^+, and all the arcs in $\mathcal{T} \setminus \{(\alpha, \beta)\}$ still constitute a spanning tree of the contracted $\mathcal{R}$ at t^+. Thus, expanding

(α, β) and including it along with all the other arcs in $\mathcal{T}$ not incident to (α, β) at time t^- does not create a cycle at time t^+. Any outer node that was adjacent to either α or β in $\mathcal{T}$ at time t^- still needs to be adjacent to α or β in $\mathcal{T}$ at time t^+. However, its neighbor may change from α to β or vice versa. We add the appropriate arcs connecting these outer nodes to (α, β) to get $\mathcal{T}$ at t^+. These changes can be done in $O(\log n)$ time using link-cut tree operations.

If instead $(\alpha, \beta) \in \mathcal{L}$ at t^-, then $(\alpha, \beta) \notin \mathcal{T}$. There exists a unique path from α to β in $\mathcal{T}$. Let ζ be an outer node incident to α lying on this path; ζ can be found in $O(\log n)$ time using a link-cut tree operation. Replacing the arc (α, ζ) by (α, β) in $\mathcal{T}$ still keeps it a spanning tree of $\mathcal{R}$, and we can now use the argument from the previous paragraph.

Now, if α and β share two arcs in $\mathcal{R}$ at time t^-, then they continue to share two arcs at time t^+ and the combinatorial structure of $\mathcal{R}$ does not change. We simply keep the same number of loop arcs between α and β.

We now discuss handling changes to level set contours. We call $\textsc{Handoff}(u, v)$ once for each arc that u represents. If any level set contours then belong to v, we call $\textsc{LSCross}(v, u)$. Finally, we call $\textsc{Handoff}(v, u)$ as many times as necessary so that both u and v have the same number of level sets as there are incident loop arcs not already represented by other vertices at time t^+.

***Other crossing event*:** We now describe how to handle non-external crossing events. Suppose we have representative vertex x and a vertex $u \neq x$ with $h(x, t) = h(u, t)$. We call $\textsc{Handoff}(u, x)$ for each loop arc represented by u followed by a single call to $\textsc{LSCross}(x, u)$ and then the same number of calls to $\textsc{Handoff}(x, u)$ as were done earlier for (u, x). In doing so, the crossing level sets of x and u retain the relative order by height and the lists of edge copies for the level sets retain their proper ordering.

***Repairing descent and ascent trees*:** During an event at time t, a constant number of edges gain or lose an intersecting crossing level set or have their endpoints change relative order by height. For each such edge $(u, v) \in \mathbb{M}$, we proactively update the descent and ascent trees of u and v in $O(\log n)$ time. We use a link-cut tree operation to remove both u and v's parents in the descent trees; if either parent was a level set root, then we delete it. Let $w \neq u$ be any down neighbor of v after the event is over (if w does not exist, v is a minimum at t^+ and hence made a root). If (v, w) does not intersect a crossing level set internally, we set w to be v's parent; else we add a new level set root c and make it v's parent by a link-cut tree operation, recording (v, w) on the arc (v, c). We perform a similar operation for u. The ascent tree is handled similarly.

We now state our main theorem.

▶ **Theorem 4.** *We can maintain the Reeb graph of a time-varying, piecewise-linear function, defined on a triangulated orientable 2-manifold without boundary of size n and genus g, in $O(\log^2 n)$ time per event and $O(gn)$ total space. The total number of events processed is $O((\kappa + g)\lambda_{O(1)}(n) \log n)$, where κ is the total number of combinatorial changes in the Reeb graph.*

Proof. There are $O(n)$ local events by our assumptions on the height function. There are g representative vertices $\mathcal{V}$ (counting with multiplicity). The set $\mathcal{V}$ changes only at a shift, birth, death, or interchange events, but these events change the combinatorial structure of the Reeb graph as well, and there are κ such events. Thus, at most $g + \kappa$ vertices can become representatives. Crossing events only occur when a representative vertex and some other vertex achieve the same height; by our assumptions on the height function, there are at most $O((\kappa + g)n)$ such events. Finally, we add and remove vertices from tournaments only

during crossing events. Therefore, there are $O((\kappa + g)\lambda_{O(1)}(n) \log n)$ tournament events [1]. Every event described above involves a constant number of link-cut tree operations, calls to LSCross and Handoff, or tournament operations. Therefore, every event, including tournament events, is handled in $O(\log^2 n)$ time [1]. ◀

Remarks. (i) Our approach can be extended to *augmented Reeb graphs*, although it does create additional types of interchange events. The augmented Reeb graph contains an additional degree two node at $\rho(v)$ for each regular v in $\mathbb{M}$. The total number of events remains the same as in the case above.
(ii) As mentioned, our approach can also handle vertices of high degree, however the running time to handle an event increases by a factor proportional to the degree of the vertices involved. Our approach can also handle non-simple saddles, without major modifications.
(iii) One of the main difficulties with extending our technique to higher dimensional manifolds is that the contours are not polygonal cycles but are higher dimensional surfaces. It is not clear how to represent them using link-cut trees, unlike the one-dimensional case.

References

1 Pankaj K. Agarwal, Haim Kaplan, and Micha Sharir. Kinetic and dynamic data structures for closest pair and all nearest neighbors. *ACM Trans. Algo.*, 5(1):4:1–4:37, 2008.

2 Pankaj K. Agarwal, Thomas Mølhave, Morten Revsbæk, Issam Safa, Yusu Wang, and Jungwoo Yang. Maintaining contour trees of dynamic terrains. In *Proc. 31st Int. Symp. Comp. Geom.*, pages 796–811, 2015.

3 Pankaj K. Agarwal and Micha Sharir. Davenport-schinzel sequences and their geometric applications. *Handbook of computational geometry*, pages 1–47, 2000.

4 Julien Basch, Leonidas J Guibas, and John Hershberger. Data structures for mobile data. *J. Algo.*, 31(1):1–28, 1999.

5 Dmitriy Bespalov, William C Regli, and Ali Shokoufandeh. Reeb graph based shape retrieval for cad. In *Proc. Int. Des. Engg. Tech. Conf. Comput. Inf. Engg.*, pages 229–238, 2003.

6 Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in Reeb graphs of 2-manifolds. In *19th Int. Symp. Comp. Geome.*, pages 344–350. ACM, 2003.

7 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong. *Computational geometry*. Springer, 3rd edition, 2000.

8 Harish Doraiswamy and Vijay Natarajan. Computing reeb graphs as a union of contour trees. *IEEE Trans. Visualiz. Comput. Graph.*, 19(2):249–262, 2013.

9 Herbert Edelsbrunner, John Harer, Ajith Mascarenhas, and Valerio Pascucci. Time-varying Reeb graphs for continuous space-time data. In *20th Int. Symp. Comp. Geome.*, pages 366–372. ACM, 2004.

10 A. T. Fomenko and T. L. Kunii. *Topological Methods for Visualization*. Springer-Verlag, Tokyo, Japan, 1997.

11 Leonidas J. Guibas. Modeling motion. In *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 1117–1134. 2004.

12 William Harvey, Yusu Wang, and Rephael Wenger. A randomized $O(m \log m)$ time algorithm for computing Reeb graphs of arbitrary simplicial complexes. In *Proc. 26th Symp. Comput. Geom.*, pages 267–276, 2010.

13 Salman Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. *Disc. Comput. Geom.*, 49(4):864–878, 2013.

14 Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

15 Georges Reeb. Sur les points singuliers d'une forme de pfaff completement intégrable ou d'une fonction numérique. *CR Acad. Sci. Paris*, 222(847-849):2, 1946.

16 Yoshihisa Shinagawa and Tosiyasu L Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Comp. Graph. App.*, 11(6):44–51, 1991.

17 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

18 B-S Sohn and Chandrajit Bajaj. Time-varying contour topology. *IEEE Trans. Visual. and Comp. Graph*, 12(1):14–25, 2006.

19 Andrzej Szymczak. Subdomain aware contour trees and contour evolution in time-dependent scalar fields. In *Inter. Conf. Shape Modeling Appli.*, pages 136–144. IEEE, 2005.

20 Tony Tung and Francis Schmitt. The augmented multiresolution reeb graph approach for content-based retrieval of 3d shapes. *Int. J. Shape Modeling*, 11(01):91–120, 2005.

On the Parameterized Complexity of Simultaneous Deletion Problems*

Akanksha Agrawal[1], R. Krithika[2], Daniel Lokshtanov[3], Amer E. Mouawad[4], and M. S. Ramanujan[5]

1 **Department of Informatics, University of Bergen, Bergen, Norway**
akanksha.agrawal@uib.no

2 **Institute of Mathematical Sciences, HBNI, Chennai, India**
rkrithika@imsc.res.in

3 **Department of Informatics, University of Bergen, Bergen, Norway**
daniello@ii.uib.no

4 **Department of Informatics, University of Bergen, Bergen, Norway**
a.mouawad@uib.no

5 **Algorithms and Complexity Group, Vienna University of Technology, Vienna, Austria**
ramanujan@ac.tuwien.ac.at

Abstract

For a family of graphs $\mathcal{F}$, an n-vertex graph G, and a positive integer k, the $\mathcal{F}$-Deletion problem asks whether we can delete at most k vertices from G to obtain a graph in $\mathcal{F}$. $\mathcal{F}$-Deletion generalizes many classical graph problems such as Vertex Cover, Feedback Vertex Set, and Odd Cycle Transversal. A (multi) graph $G = (V, \cup_{i=1}^{\alpha} E_i)$, where the edge set of G is partitioned into α color classes, is called an α-edge-colored graph. A natural extension of the $\mathcal{F}$-Deletion problem to edge-colored graphs is the Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion problem. In the latter problem, we are given an α-edge-colored graph G and the goal is to find a set S of at most k vertices such that each graph $G_i - S$, where $G_i = (V, E_i)$ and $1 \leq i \leq \alpha$, is in $\mathcal{F}_i$. Recently, a subset of the authors considered the aforementioned problem with $\mathcal{F}_1 = \ldots = \mathcal{F}_\alpha$ being the family of all forests. They showed that the problem is fixed-parameter tractable when parameterized by k and α and can be solved in $\mathcal{O}^\star(2^{\mathcal{O}(\alpha k)})$ time[1]. In this work, we initiate the investigation of the complexity of Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion with different families of graphs. In the process, we obtain a complete characterization of the parameterized complexity of this problem when one or more of the $\mathcal{F}_i's$ is the class of bipartite graphs and the rest (if any) are forests. We show that if $\mathcal{F}_1$ is the family of all bipartite graphs and each of $\mathcal{F}_2 = \mathcal{F}_3 = \ldots = \mathcal{F}_\alpha$ is the family of all forests then the problem is fixed-parameter tractable parameterized by k and α. However, even when $\mathcal{F}_1$ and $\mathcal{F}_2$ are both the family of all bipartite graphs, then the Simultaneous $(\mathcal{F}_1, \mathcal{F}_2)$-Deletion problem itself is already W[1]-hard.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases parameterized complexity, feedback vertex set, odd cycle transversal, edge-colored graphs, simultaneous deletion

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.9

* Due to space limitations most proofs have been omitted.
[1] We use the $\mathcal{O}^\star$ notation which hides factors that are polynomial in the input size.

1 Introduction

Given their tremendous modelling power, graphs have become an integral part of theoretical computer science in general, and of algorithm design in particular. One graph problem which encapsulates many problems of both practical and theoretical interest is $\mathcal{F}$-Deletion. For a family of graphs $\mathcal{F}$, an n-vertex graph G, and a positive integer k, the $\mathcal{F}$-Deletion problem asks whether we can delete at most k vertices from G to obtain a graph in $\mathcal{F}$. To state a few, $\mathcal{F}$-Deletion generalizes problems such as Vertex Cover [6], Feedback Vertex Set (FVS) [5, 8, 17], Vertex Planarization [15], Odd Cycle Transversal (OCT) [22, 14, 18], Interval Vertex Deletion [3], Chordal Vertex Deletion [4], and Planar $\mathcal{F}$-Deletion [11, 16].

A graph $G = (V, \cup_{i=1}^{\alpha} E_i)$, where the edge set of G is partitioned into α color classes, is called an α-edge-colored graph. Edge-colored graphs are fundamental in graph theory and have been extensively studied in the literature, especially for alternating cycles and monochromatic subgraphs [2]. A natural extension of the $\mathcal{F}$-Deletion problem to edge-colored graphs is the Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion problem. In the latter problem, we are given an α-edge-colored graph G and the goal is to find a set S of at most k vertices such that each graph $G_i - S$ is in $\mathcal{F}_i$, where $G_i = (V, E_i)$ and $1 \leq i \leq \alpha$. Recently, Cai and Ye [2] studied several problems restricted to 2-edge-colored graphs, where edges are colored either red or blue. They asked, as an open question, whether the Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion problem parameterized by k, with $\alpha = 2$ and $\mathcal{F}_1 = \mathcal{F}_2$ being the family of all forests, is fixed-parameter tractable (FPT), i.e. whether the problem can be solved in $\mathcal{O}^\star(f(k))$ time [10] (for some computable function f). Agrawal et al. [1] and Ye [24] answered this question in the affirmative. In particular, it was shown in [1] that the problem can be solved by an algorithm running in $\mathcal{O}^\star(2^{\mathcal{O}(\alpha k)})$ time. This work pointed to a few natural further directions for research. For instance, does Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion remain fixed-parameter tractable when the family of all forests is replaced by the family of all bipartite graphs? What is the complexity of the problem when not all families are equal?

The results in this work allow us to take a significant step towards a better understanding of simultaneous deletion problems in general. To that end, we investigate the complexity of Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion in two settings. First, we consider the problem with $\mathcal{F}_1$ being the family of all bipartite graphs and $\mathcal{F}_2 = \mathcal{F}_3 = \ldots = \mathcal{F}_\alpha$ being the family of all forests. We call this problem Simultaneous FVS/OCT and define it as follows.

Simultaneous FVS/OCT	**Parameter(s):** k and α
Input: An α-edge-colored graph $G = (V, \cup_{i=1}^{\alpha} E_i)$ and an integer k.	
Question: Is there a set $S \subseteq V$ of size at most k such that $G_1 - S$ is a bipartite graph and $G_2 - S$, …, $G_\alpha - S$ are acyclic, where $G_i = (V, E_i)$ and $1 \leq i \leq \alpha$?	

We call a solution S to the Simultaneous FVS/OCT problem a *sim-fvs-oct*. Our first contribution is an algorithm that, given an instance $(G = (V, \cup_{i=1}^{\alpha} E_i), k)$ of Simultaneous FVS/OCT, runs in time $\mathcal{O}^\star(k^{\mathsf{poly}(\alpha,k)})$ and either computes a sim-fvs-oct in G of size at most k or correctly concludes that such a set does not exist.

In the second setting, we consider the Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion problem where $\mathcal{F}_1 = \ldots = \mathcal{F}_\alpha$ is the family of all bipartite graphs. We call this problem Simultaneous OCT and define it as follows.

Simultaneous OCT	**Parameter(s):** k and α
Input: An α-edge-colored graph $G = (V, \cup_{i=1}^{\alpha} E_i)$ and an integer k.	
Question: Is there a set $S \subseteq V$ of size at most k such that $G_i - S$ is bipartite, where $G_i = (V, E_i)$ and $1 \leq i \leq \alpha$?	

We refer to a solution S to the Simultaneous OCT problem as a *sim-oct*. Our second (and rather surprising) contribution is a negative answer to the first open question of Agrawal et al. [1]. We show that, even for $\alpha = 2$, the Simultaneous OCT problem is W[1]-hard. To prove this result, we first reduce the well-known Multicolored Clique problem [7] to an auxiliary problem we call Simultaneous Cut. Simultaneous Cut is a natural generalization of the classical (s,t)-CUT problem to edge-colored graphs. Finally, we show that Simultaneous Cut can be reduced to Simultaneous OCT. Notice that W[1]-hardness of Simultaneous OCT implies that Simultaneous $(\mathcal{F}_1, \ldots, \mathcal{F}_\alpha)$-Deletion problem with at least two of the families being the family of all bipartite graphs is W[1]-hard.

Overview of the algorithm. Note that for any fixed k and α, our algorithm for solving the Simultaneous FVS/OCT problem runs in polynomial time. The said algorithm can be broken down into four stages, three of which are reductions to auxiliary problems. Initially, as was first proposed by Ye [24], we use the notion of compact representations of feedback vertex sets (see Section 2 for formal definitions) to reduce Simultaneous FVS/OCT into $2^{\mathcal{O}(\alpha k)}$ instances of the Colorful OCT problem, which is formally defined as follows. We note that, in any reduced instance, ℓ will be bounded from above by αk.

Colorful OCT	**Parameter(s):** k and ℓ
Input: A graph $G = (V, E)$, integers k and ℓ, and a grouping $\mathcal{P}$ of the vertices of G into (not necessarily distinct) sets $\{P_1, \ldots, P_\ell\}$.	
Question: Is there a set $S \subseteq V$ of size at most k such that $G - S$ is a bipartite graph and $S \cap P_i \neq \emptyset$, for $i \in \{1, \ldots, \ell\}$?	

Intuitively, compact representations give us a partition of a vertex subset of the graph into sets such that picking one vertex from each part is "guaranteed" to constitute a feedback vertex set of each graph G_i, $2 \leq i \leq \alpha$. As such, we are able to encode the feedback vertex set "side" of the Simultaneous FVS/OCT problem (via the reduction) as colors on the vertices (i.e. different sets in $\mathcal{P}$ represent different colors for each vertex) and focus on a "colored" variant of Odd Cycle Transversal. Naturally, the second stage is to solve the Colorful OCT problem within the claimed running time. To do so, we reduce an instance of Colorful OCT to an instance of the compression variant of the problem, i.e. Colorful OCT Compression. This problem assumes an odd cycle transversal of size at most k as part of the input. Note that finding an odd cycle transversal of a graph $G = (V, E)$ of size at most k can be accomplished using the fixed-parameter tractable algorithms for OCT parameterized by solution size [14, 22], both of which run in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time.

Colorful OCT Compression	**Parameter(s):** k and ℓ
Input: A graph $G = (V, E)$, integers k and ℓ, a grouping $\mathcal{P}$ of the vertices of G into (not necessarily distinct) sets $\{P_1, \ldots, P_\ell\}$, and a set $O \subseteq V(G)$ of size at most k such that $G - O$ is bipartite.	
Question: Is there a set $S \subseteq V$ of size at most k such that $G - S$ is a bipartite graph and $S \cap P_i \neq \emptyset$, for $i \in \{1, \ldots, \ell\}$?	

Now, to solve an instance of Colorful OCT Compression, we reduce it into $2^{\mathcal{O}(k)}$ instances of yet another problem, namely Colorful Separator. This reduction is in

many ways similar to the iterative compression algorithm for solving the Odd Cycle Transversal problem [7, 13, 23].

Colorful Separator	**Parameter(s):** k and ℓ
Input: A graph $G = (V, E)$, integers k and ℓ, a grouping $\mathcal{P}$ of the vertices of G into (not necessarily distinct) sets $\{P_1, \ldots, P_\ell\}$, and vertices s and t in $V(G)$.	
Question: Is there an (s,t)-separator $S \subseteq V \setminus \{s, t\}$ such that $\lvert S\rvert \leq k$ and $S \cap P_i \neq \emptyset$, for each $i \in \{1, \ldots, \ell\}$?	

Finally, and arguably the most technical part of our algorithm, is to show how to solve an instance of Colorful Separator. We will in fact solve a much more general problem, which we define in Section 4 (to keep the presentation clear). Our two main ingredients are a dynamic programming routine and a generalization of the concept of important separators, which has been recently defined to design parameterized algorithms for several "cut" problems [12, 19, 20]. We note that an alternative algorithm for solving Colorful Separator can be obtained by applying the treewidth reduction result of Marx et al. [21]. However, a "simple" application of this result would give an algorithm with a worse running time (double exponential).

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N}$. For $n \in \mathbb{N}$, we let $[n]$ denote the set $\{1, 2, \ldots, n\}$. Given a universe $\mathcal{U}$, a set $S \subseteq \mathcal{U}$, and a family of sets $\mathcal{F} = \{F_1, \ldots, F_\ell\}$ over $\mathcal{U}$, we let $\mathcal{F}\big|_S$ denote the *restriction* of F to S, i.e. $\mathcal{F}\big|_S = \{F_1 \cap S, \ldots, F_\ell \cap S\}$. We use standard terminology from the book of Diestel [9] for the graph-related terms which are not explicitly defined here. For a graph G, we use $V(G)$ and $E(G)$ to denote the vertex and edge sets of G, respectively. For $S \subseteq V(G)$, by $N_G(S)$ we denote the set $\{u \in V(G) \setminus S \mid (u, v) \in E(G) \wedge v \in S\}$. We drop the subscript G from $N_G(S)$ when the context is clear. For a vertex subset $S \subseteq V(G)$, by $G[S]$ we denote the graph with vertex set S and edge set $\{(u, v) \in E(G) \mid u, v \in S\}$. By $G - S$ we denote the graph $G[V(G) \setminus S]$. For $X, Y \subseteq V(G)$, an (X, Y)-path in G is a path $v_1, v_2, \ldots, v_\ell$ such that $v_1 \in X$ and $v_\ell \in Y$. We say that X and Y are *linked* in G if there exists an (X, Y)-path in G. We say that vertices in Y are *reachable* from X if, for all $y \in Y$, there exists $x \in X$ such that there is a path from x to y.

A vertex subset $S \subseteq V(G)$ is a *feedback vertex set (fvs)* in G if $G - S$ is a forest. If there is no $S' \subset S$ such that $G - S'$ is a forest then S is a *minimal feedback vertex set (minimal fvs)* in G. A vertex subset $S \subseteq V(G)$ is an *odd cycle transversal (oct)* in G if $G - S$ is bipartite. If there is no $S' \subset S$ such that $G - S'$ is a bipartite graph then S is a *minimal odd cycle transversal (minimal oct)* in G. For a graph G and set $X \subseteq V(G)$, we refer to a partition (A, B) of X as a *valid bipartition* of $G[X]$ if $G[A]$ and $G[B]$ are both edgeless graphs. We refer to a valid bipartition of $V(G)$ as a valid bipartition of the graph G.

▶ **Definition 2.1.** Let G be a graph and X and Y be disjoint subsets of $V(G)$. A vertex set S disjoint from $X \cup Y$ is called an *(X,Y)-separator* if there is no (X, Y)-path in $G - S$. We denote by $R_G(X, S)$ the set of vertices of $G - S$ reachable from vertices of X via paths and by $NR_G(X, S)$ the set of vertices of $G - S$ *not* reachable from vertices of X.

▶ **Definition 2.2.** [13] A *compact representation* of a set $\mathcal{S}$ of minimal feedback vertex sets of a graph G is a collection $\mathcal{C}$ of pairwise disjoint subsets of $V(G)$ such that choosing exactly one vertex from every set in $\mathcal{C}$ results in a minimal feedback vertex set for G that is in $\mathcal{S}$.

▶ **Lemma 2.3.** [13] *The set of all minimal feedback vertex sets of size at most k can be represented by a collection of compact representations of size $2^{\mathcal{O}(k)}$. Furthermore, given a*

graph $G = (V, E)$ and a feedback vertex set F for G of size $k + 1$, we can enumerate the compact representations of all minimal feedback vertex sets for G having size at most k in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time.

3 From Simultaneous FVS/OCT to Colorful OCT

We first describe how to reduce an instance of Simultaneous FVS/OCT to $2^{\mathcal{O}(\alpha k)}$ instances of Colorful OCT. Note that since both Feedback Vertex Set [20] and Odd Cycle Transversal [22, 14] can be solved in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time, we assume that, along with an instance $(G = (V, \cup_{i=1}^{\alpha} E_i), k)$, we are given sets $O, F_2, \ldots, F_\alpha \subseteq V(G)$ of size at most k such that $G_1 - O$ is a bipartite graph and $G_i - F_i$, $2 \leq i \leq \alpha$, is acyclic (as otherwise we can safely conclude that the given instance is a no-instance).

▶ **Lemma 3.1.** *There is an algorithm that, given an instance $(G = (V, \cup_{i=1}^{\alpha} E_i), k)$ of* Simultaneous FVS/OCT*, runs in time $\mathcal{O}^\star(2^{\mathcal{O}(\alpha k)})$ and returns a set of $2^{\mathcal{O}(\alpha k)}$ instances of* Colorful OCT *such that the original instance is a yes-instance if and only if at least one of the returned instances is a yes-instance.*

Proof. Armed with the sets F_i which are of size at most k, we apply the algorithm of Lemma 2.3 to each graph G_i, $2 \leq i \leq \alpha$, to obtain a set of compact representations $\mathbf{C}_i = \{\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots\}$, $2 \leq i \leq \alpha$. Note that each $\mathbf{C}_i$ is of size $2^{\mathcal{O}(k)}$ and each $\mathcal{C}_i^j$ is of size at most k. The said algorithm runs in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time for each graph G_i. For each tuple $\{\mathcal{C}_2^{j_2}, \ldots, \mathcal{C}_\alpha^{j_\alpha}\} \in \mathbf{C}_2 \times \ldots \times \mathbf{C}_\alpha$, we construct an instance $(G', \mathcal{P}, k', \ell)$ of Colorful OCT as follows. We let $G' = (V, E_1)$, $k' = k$, and $\ell = \sum_{i=2}^{\alpha} |\mathcal{C}_i^j| \leq \alpha k$.

For each $\mathcal{C} \in \{\mathcal{C}_2^{j_2}, \ldots, \mathcal{C}_\alpha^{j_\alpha}\}$ and for each set $C \in \mathcal{C}$, we add a set $P \in \mathcal{P}$ and we let $P = C$. In other words, all vertices in C are added to P. Observe that $|\mathcal{C}| \leq k$. Since each $\mathbf{C}_i$ is of size $2^{\mathcal{O}(k)}$, it is easy to verify that the number of instances is in fact $2^{\mathcal{O}(\alpha k)}$. We now prove the correctness of the algorithm.

Assume that $(G = (V, \cup_{i=1}^{\alpha} E_i), k)$ is a yes-instance and let S be a solution of size at most k. Note that S need not be a minimal fvs in G_i, $2 \leq i \leq \alpha$. However, for each $i \in \{2, \ldots, \alpha\}$, there exists a set $S' \subseteq S$ such that S' is a minimal fvs for G_i. Hence, by Definition 2.2 and Lemma 2.3, for every $i \in \{2, \ldots, \alpha\}$, there exists a $\mathcal{C}_i^j \in \mathbf{C}_i$ such that for all $C \in \mathcal{C}_i^j$ we have $S' \cap C \neq \emptyset$. Since we enumerate all compact representations and create one instance for each, we know that at least one instance $(G', \mathcal{P}, k', \ell)$ of Colorful OCT will correspond to the correct choice. The fact that S is a solution for $(G', \mathcal{P}, k', \ell)$ follows from the fact that S contains a minimal oct for G_1.

For the other direction, let S' be a solution for an instance $(G', \mathcal{P}, k', \ell)$ of Colorful OCT. Since S' is of size at most k, it is clearly an oct for G_1. Moreover, since S' must intersect every $P \in \mathcal{P}$, it follows from the definition of compact representations and our construction that S' is an fvs for G_i, $2 \leq i \leq \alpha$, as needed. ◀

We now focus on solving an instance $(G, \mathcal{P}, k, \ell)$ of Colorful OCT. Recall that we also have access to the set O which is an oct of G of size at most k. Our next step is to reduce $(G, \mathcal{P}, k, \ell)$ to an instance $(G, \mathcal{P}, O, k, \ell)$ of Colorful OCT Compression. The correctness of this reduction is immediate. The final piece in our sequence of reductions is to reduce $(G, \mathcal{P}, O, k, \ell)$ to $2^{\mathcal{O}(k)}$ instances of Colorful Separator.

▶ **Lemma 3.2.** *There is an algorithm that, given an instance $(G, \mathcal{P}, O, k, \ell)$ of* Colorful OCT Compression*, runs in time $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ and returns a set of $2^{\mathcal{O}(k)}$ instances of* Colorful Separator *such that the original instance is a yes-instance if and only if at least one of the returned instances is a yes-instance.*

To summarize, given an instance $(G = (V, \cup_{i=1}^{\alpha} E_i), k)$ of Simultaneous FVS/OCT, we first compute an odd cycle transversal of G_1 and a feedback vertex set of G_i, $i \in [\alpha] \setminus \{1\}$, in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time. Then, we generate $2^{\mathcal{O}(\alpha k)}$ instances of Colorful OCT, of the form $(G, \mathcal{P}, k, \ell \leq \alpha k)$, in $\mathcal{O}^\star(2^{\mathcal{O}(\alpha k)})$ time. Each instance of Colorful OCT is converted into an instance $(G, \mathcal{P}, O, k, \ell)$ of Colorful OCT Compression in polynomial time. Finally, for each instance of Colorful OCT Compression we generate $2^{\mathcal{O}(k)}$ instances of Colorful Separator, with parameters k and $\ell \leq \alpha k$, in $\mathcal{O}^\star(2^{\mathcal{O}(k)})$ time. Lemmas 3.1 and 3.2 together imply that if we can solve an instance of Colorful Separator in $\mathcal{O}^\star(k^{\mathsf{poly}(\alpha,k)})$ time then the algorithm for Simultaneous FVS/OCT follows.

4 An FPT algorithm for finding colorful separators

We in fact give an algorithm for a more general problem, which we call Colorful Multiway Cut (or CMWC for short). Before we proceed, we need a few definitions.

▶ **Definition 4.1.** Given a graph G, a set $T \subseteq V(G)$, and a partition $\mathcal{T}$ of T into (pairwise disjoint) sets $\{T_1, \ldots, T_r\}$, we say that $S \subseteq V(G) \setminus T$ is a *$\mathcal{T}$-multiway cut* if, in $G - S$, no vertex in $T_i \setminus S$ can reach a vertex in $T_j \setminus S$, for all $i, j \in [r]$, such that $i \neq j$. We say that $\mathcal{T}$ is an *edge-free partition* of T if there are no edges (u, v) in $G[T]$ where u and v belong to different sets of $\mathcal{T}$.

Given a grouping $\{P_1, \ldots, P_\ell\}$ of the vertices of a graph G, we define a partial coloring function $\mathsf{col} : V(G) \to 2^{[\ell]}$. That is, we have $i \in \mathsf{col}(v)$ if and only if $v \in P_i$, for some $i \in [\ell]$. In this context, for a set $C \subseteq [\ell]$, a subset S of vertices of G is called *C-colorful* if, for each $i \in C$, there is a vertex v in S such that $i \in \mathsf{col}(v)$. For a subset $S \subseteq V(G)$, we denote by $\mathsf{col}(S)$ the set $\{j \mid v \in S \cap (\bigcup_{i=1}^{\ell} P_i) \wedge j \in \mathsf{col}(v)\}$, i.e. the set of colors appearing in S. The CMWC can now be defined as follows.

Colorful Multiway Cut (CMWC) **Parameter(s):** k, $\lvert T\rvert$, and ℓ
Input: A graph $G = (V, E)$, a set $T \subseteq V(G)$, a partition $\mathcal{T}$ of T into (pairwise disjoint) sets $\{T_1, \ldots, T_r\}$, a grouping $\mathcal{P}$ of the vertices of G into (not necessarily distinct) sets $\{P_1, \ldots, P_\ell\}$, a set $C \subseteq [\ell]$, and an integer k.
Question: Is there a set $S \subseteq V(G) \setminus T$ such that $\lvert S\rvert \leq k$, S is a $\mathcal{T}$-multiway cut in G, and S is C-colorful?

4.1 Setting up the algorithm

Let $(G, T, \mathcal{T}, \mathcal{P}, C, k)$ be an instance of CMWC. We start by stating a few simple reduction rules (which are applied in the order they are stated).

▶ **Reduction Rule 1.** *If $k < 0$ then return* false*, i.e. $(G, T, \mathcal{T}, \mathcal{P}, C, k)$ is a no-instance.*

▶ **Reduction Rule 2.** *If $k = 0$ and $\emptyset$ is a solution to $(G, T, \mathcal{T}, \mathcal{P}, C, k)$ then return* true*, i.e. $(G, T, \mathcal{T}, \mathcal{P}, C, k)$ is yes-instance. If $k = 0$ and $\emptyset$ is not a solution then return* false*.*

▶ **Reduction Rule 3.** *If there exists $i \in C$ such that $P_i \subseteq T$ then return* false*.*

▶ **Reduction Rule 4.** *If there exists $i \in C$ such that $P_i \cap T \neq \emptyset$ then set $P_i = P_i \setminus T$.*

▶ **Reduction Rule 5.** *If there exists $i \in C$ such that $P_i = \emptyset$ then return* false*.*

▶ **Reduction Rule 6.** *If $\mathcal{T}$ is not an edge-free partition then return* false*.*

It is easy to see that Reduction Rules 1 to 6 are safe and can be applied in polynomial time. When $k > 0$ and $\emptyset$ is a $\mathcal{T}$-multiway cut, we can solve the corresponding instance in time $\mathcal{O}^\star(2^{\mathcal{O}(\ell)})$. The following observation describes how.

▶ **Observation 1.** *Let $\mathcal{I} = (G, T, \mathcal{T}, \mathcal{P}, C, k)$ be an instance of* Colorful Multiway Cut. *If $k > 0$ and $\emptyset$ is a $\mathcal{T}$-multiway cut then $\mathcal{I}$ can be solved in $\mathcal{O}(2^{\mathcal{O}(\ell)}n^2)$ time, where $n = |V(G)|$.*

Proof. If $k > 0$ and $\emptyset$ is a $\mathcal{T}$-multiway cut then we are left with the problem of finding a set $S \subseteq V(G) \setminus T$ of size at most k such that $S \cap P_i \neq \emptyset$, for each $i \in C$. Hence, we construct a family $\mathcal{F}$ consisting of a set $f_{P_i} = P_i$ for each for each $i \in C$ and we let $\mathcal{U} = \cup_{i \in C} P_i$. Note that $|\mathcal{F}| \leq \ell \leq \alpha k$ and $|\mathcal{U}| \leq |V(G)|$. Since Reduction Rules 3, 4, and 5 are not applicable, for each $i \in C$, we have $f_{P_i} \neq \emptyset$ and $P_i \cap T = \emptyset$. If we can find a subset $U \subseteq \mathcal{U}$ which intersects all the sets in $\mathcal{F}$, such that $|U| \leq k$, then U is the required solution. Otherwise, we have a no-instance. It is known that the Hitting Set problem parameterized by the size of the family $\mathcal{F}$ is fixed-parameter tractable and can be solved in $\mathcal{O}(2^{\mathcal{O}(|\mathcal{F}|)}|\mathcal{U}|^2)$ time [7]. In particular, we can find an optimum hitting set $U \subseteq \mathcal{U}$, hitting all the sets in $\mathcal{F}$. Therefore, we have a subset of vertices that intersects all sets P_i, for $i \in C$. ◀

Before proceeding with the description of the algorithm, we first recall the notion of tight separator sequences introduced in [19]. However, the definition and structural lemmas regarding tight separator sequences used in this paper are from [20]. Note that although [20] contains Definition 4.2 and Lemma 4.3 in terms of directed graphs, the same holds true for undirected graphs because one can represent any undirected graph as a directed graph by adding bidirectional edges between every pair of adjacent vertices.

▶ **Definition 4.2.** Let X and Y be two subsets of $V(G)$ and let $k \in \mathbb{N}$. A *tight (X,Y)-reachability sequence* of order k is an ordered collection $\mathcal{H} = \{H_0, H_1, \ldots, H_q, H_{q+1}\}$ of sets in $V(G)$ satisfying the following properties:

- $X \subseteq H_i \subseteq V(G) \setminus N[Y]$ for any $0 \leq i \leq q$;
- $X = H_0 \subset H_1 \subset H_2 \subset \ldots \subset H_q \subset H_{q+1} = V(G) \setminus Y$;
- H_i is reachable from X in $G[H_i]$ and every vertex in $N(H_i)$ can reach Y in $G - H_i$ (implying that $N(H_i)$ is a minimal (X,Y)-separator in G);
- $|N(H_i)| \leq k$ for every $1 \leq i \leq q$;
- $N(H_i) \cap N(H_j) = \emptyset$ for all $1 \leq i, j \leq q$ and $i \neq j$;
- For any $0 \leq i \leq q-1$, there is no (X,Y)-separator S of size at most k where $S \subseteq H_{i+1} \setminus N[H_i]$ or $S \cap N[H_q] = \emptyset$ or $S \subseteq H_1$.

We let $Q_0 = X$, $Q_i = N(H_i)$, for $1 \leq i \leq q$, $Q_{q+1} = Y$, and $\mathcal{Q} = \{Q_0, Q_1, \ldots, Q_q, Q_{q+1}\}$. We call $\mathcal{Q}$ a *tight (X,Y)-separator sequence* of order k.

▶ **Lemma 4.3.** *(see [20]) There is an algorithm that, given a graph G on n vertices and m edges, subsets $X, Y \subseteq V(G)$ and $k \in \mathbb{N}$, runs in time $\mathcal{O}(k^2nm)$ and either correctly concludes that there is no (X,Y)-separator of size at most k in G or returns the sets $H_0, H_1, H_2 \setminus H_1, \ldots, H_q \setminus H_{q-1}, H_{q+1} \setminus H_q$ corresponding to a tight (X,Y)-reachability sequence $\mathcal{H} = \{H_0, H_1, \ldots, H_q, H_{q+1}\}$ of order k.*

Our algorithm will be a combination of dynamic programming over the sets Q_i, $0 \leq i \leq q+1$, and recursive calls for solving "smaller" instances of the same problem. Below we state some observations that help understand the structure of a solution and are crucial for achieving the stated running time.

Algorithm 1: Pseudocode for **ALG1**

Input: $(G, T, \mathcal{T}, \mathcal{P}, C, k)$
Output: true or false

1 Apply all reduction rules (in order) and return true/false appropriately (if applicable).
2 **if** $k > 0$ *and* $\emptyset$ *is a* $\mathcal{T}$*-multiway cut* **then**
3 **return** true/false appropriately (Observation 1)
4 Let $T_1 \in \mathcal{T}$ such that T_1 is linked to some $T_j \in \mathcal{T}$, where $j \neq 1$.
5 Let $\mathcal{H} = \{H_0, H_1, \ldots, H_q, H_{q+1}\}$ be a $(T_1, T \setminus T_1)$-reachability sequence of order k;
6 Let $\mathcal{Q} = \{Q_0, Q_1, \ldots, Q_q, Q_{q+1}\}$ be the corresponding $(T_1, T \setminus T_1)$-separator sequence;
7 **if** $\mathcal{Q} = \emptyset$ **then**
8 **return** false;
9 **return ALG2**$(G, T, \mathcal{T}, \mathcal{P}, C, k, \mathcal{Q})$;

▶ **Observation 2.** *Let* $(G, T, \mathcal{T}, \mathcal{P}, C, k)$ *be an instance of* Colorful Multiway Cut *and let* T_1 *be a set in* $\mathcal{T}$ *which is linked to some set in* $\mathcal{T} \setminus \{T_1\}$*. Moreover, let* $\mathcal{H} = \{H_0, H_1, \ldots, H_q, H_{q+1}\}$ *be a tight* $(T_1, T \setminus T_1)$*-reachability sequence of order* k *and let* $\mathcal{Q} = \{Q_0, Q_1, \ldots, Q_q, Q_{q+1}\}$ *be the corresponding tight separator sequence. Assume* $(G, T, \mathcal{T}, \mathcal{P}, C, k)$ *is a yes-instance and let* S *be one of its solution. Then,* S *can be partitioned into the following (pairwise-disjoint) sets.*

- $Z_1 = S \cap (H_1 \setminus Q_0)$.
- $S_i = S \cap Q_i$ *for* $1 \leq i \leq q$.
- $Z_i = (S \cap (H_i \setminus N[H_{i-1}])) \setminus Q_{q+1}$ *for* $2 \leq i \leq q+1$.

▶ **Observation 3.** $|Z_i| \leq k - 1$ *for each* $i \in [q+1]$.

To keep the presentation clean, we shall define two routines **ALG1** and **ALG2**. **ALG1** (Algorithm 1) delegates most of the "heavy lifting" to **ALG2**. That is, **ALG1** simply checks if any of the reduction rules are applicable and solves the instance if it corresponds to one of the base cases. When this is not the case, **ALG1** proceeds by computing a tight separator sequence and calls **ALG2**. Note that we can safely return false when the algorithm fails to construct such a sequence (Lines 7 and 8 of Algorithm 1). We now move to the description of **ALG2**, which takes as additional input the newly constructed tight separator sequence. Roughly speaking, **ALG2** will recursively solve a "large" number of instances restricted to graphs that "reside" between two consecutive separators of a separator sequence. The number of instances will be bounded by the number of possible "interactions" between the two consecutive separators and a hypothetical solution. However, due to Observation 3, each one of those recursive calls can be made with a strictly smaller value of k. Having solved all such instances (and stored the outcomes in tables), **ALG2** then proceeds using a dynamic programming routine which computes the answer in a left-to-right manner, i.e. starting from Q_0 all the way to Q_{q+1}. We now give a formal description.

▶ **Definition 4.4.** For a graph G and a tight separator sequence $\mathcal{Q} = \{Q_0, Q_1, \ldots, Q_q, Q_{q+1}\}$, we let $G_i = G - R_G(Q_{q+1}, Q_i)$, *i.e.* the graph obtained after removing the vertices that are reachable from Q_{q+1} after deleting Q_i, and we let $\widehat{G}_i = G_i - (V(G_{i-1}) \setminus Q_{i-1})$.

For each graph G_i, $i \in [q+1]$, we maintain a table Γ_i, where each entry is indexed by a tuple $(X, \mathcal{A}, \overline{C}, \overline{p})$. For each graph $\widehat{G}_i$, $i \in [q+1]$, we maintain a table Λ_i, where each entry is indexed by a tuple $(L, R, \mathcal{B}, \widehat{C}, \widehat{p})$. The tuples are described below.

Algorithm 2: Pseudocode for **ALG2**

Input: $(G, T, \mathcal{T}, \mathcal{P}, C, k, \mathcal{Q})$
Output: true or false

1 Initialize all entries in Γ_i to false, for $i \in [q+1]$;
2 Initialize all entries in Λ_i to false, for $i \in [q+1]$;
3 **for** *each* $\widehat{G}_i \in \{\widehat{G}_1, \ldots, \widehat{G}_{q+1}\}$ **do**
4 **for** *each* $L \subseteq Q_{i-1} \setminus T$ *and each each* $R \subseteq Q_i \setminus T$ **do**
5 **for** *each edge-free partition* $\mathcal{B}$ *of* $(Q_{i-1} \cup Q_i) \setminus (L \cup R)$ **do**
6 **for** *each* $\widehat{C} \subseteq [\ell]$ *and each* $0 \leq \widehat{p} \leq k - \max\{1, |L \cup R|\}$ **do**
7 $\mathbb{I} = (\widehat{G}_i - (L \cup R), (Q_{i-1} \cup Q_i) \setminus (L \cup R), \mathcal{B}, \mathcal{P}|_{V(G_i - (L \cup R))}, \widehat{p})$;
8 $\Lambda_i(L, R, \mathcal{B}, \widehat{C}, \widehat{p}) = \mathbf{ALG1}(\mathbb{I})$;
9 Copy table entries for Γ_1, *i.e.* $\Gamma_1(X, \mathcal{A}, \overline{C}, \overline{p}) = \Lambda_1(\emptyset, X, \mathcal{A}, \overline{C}, \overline{p})$;
10 **for** *each* $G_i \in \{G_2, \ldots, G_{q+1}\}$ *(in order)* **do**
11 **for** *each* $X \subseteq Q_i \setminus T$ **do**
12 **for** *each edge-free partition* $\mathcal{A}$ *of* $(Q_i \cup Q_0) \setminus X$ **do**
13 **for** *each* $\overline{C} \subseteq [\ell]$ *and each* $0 \leq \overline{p} \leq k - |X|$ **do**
14 $\tau_1 = (X, \mathcal{A}, \overline{C}, \overline{p})$;
15 **for** *each tuple* $\tau_2 = (L, R, \mathcal{B}, \widehat{C}, \widehat{p}) \in \Lambda_i$ **do**
16 **for** *each tuple* $\tau_3 = (X', \mathcal{A}', \overline{C}', \overline{p}') \in \Gamma_{i-1}$ **do**
17 **if** τ_1, τ_2, *and* τ_3 *are compatible* **then**
18 $\Gamma_i(\tau_1) = \Gamma_i(\tau_1) \vee [\Gamma_{i-1}(\tau_3) \wedge \Lambda_i(\tau_2)]$;
19 **if** $\Gamma_{q+1}(\emptyset, \mathcal{T}, C, p) =$ *true (for some* $p \leq k$*)* **then**
20 **return** true;
21 **return** false;

- $X \subseteq Q_i \setminus T$ and $L \subseteq Q_{i-1} \setminus T$ and $R \subseteq Q_i \setminus T$;
- $\mathcal{A}$ is an edge-free partition of $(Q_i \cup Q_0) \setminus X$;
- $\mathcal{B}$ is an edge-free partition of $(Q_{i-1} \cup Q_i) \setminus (L \cup R)$;
- $\overline{C}, \widehat{C} \subseteq [\ell]$ and $\overline{p} \leq k - |X|$ and $\widehat{p} \leq k - |L \cup R|$ if $L \cup R \neq \emptyset$ and $\widehat{p} \leq k - 1$, otherwise.

▶ **Definition 4.5.** For a tuple $\tau = (X, \mathcal{A}, \overline{C}, \overline{p})$, we denote by $\mathbb{I}_\tau$ the instance $(G_i - X, (Q_i \cup Q_0) \setminus X, \mathcal{A}, \mathcal{P}|_{V(G_i - X)}, \overline{C}, \overline{p})$ of CMWC. Similarly, for a tuple $\tau = (L, R, \mathcal{B}, \widehat{C}, \widehat{p})$, we denote by $\mathbb{I}_\tau$ the instance $(\widehat{G}_i - (L \cup R), (Q_{i-1} \cup Q_i) \setminus (L \cup R), \mathcal{B}, \mathcal{P}|_{V(G_i - (L \cup R))}, \widehat{C}, \widehat{p})$ of CMWC. Finally, we define $\Gamma_i(\tau)$ (or $\Lambda_i(\tau)$)= true if and only if $\mathbb{I}_\tau$ is a yes-instance of CMWC.

▶ **Definition 4.6.** Given three tuples $\tau_1 = (X, \mathcal{A}, \overline{C}, \overline{p})$, $\tau_2 = (L, R, \mathcal{B}, \widehat{C}, \widehat{p})$, and $\tau_3 = (X', \mathcal{A}', \overline{C}', \overline{p}')$, we say that they are *compatible* if all of the following conditions hold.

- $\tau_1 \in \Gamma_i$ and $\tau_2 \in \Lambda_i$ and $\tau_3 \in \Gamma_{i-1}$, where $i \in [q+1]$;
- $X' = L$ and $X = R$;
- $\mathcal{A}|_{Q_i \setminus X} = \mathcal{B}|_{Q_i \setminus R}$ and $\mathcal{B}|_{Q_{i-1} \setminus L} = \mathcal{A}'|_{Q_{i-1} \setminus X'}$ and $\mathcal{A}|_{Q_0} = \mathcal{A}'|_{Q_0}$;
- $\overline{p}' + \widehat{p} + |L| \leq p$ and $\overline{C}' \cup \widehat{C} \cup \mathsf{col}(L) = \overline{C}$.

The complete description of **ALG2** is given in Algorithm 2. Initially, we set all table entries to false (Lines 1 and 2). Then, for each $\widehat{G}_i \in \{\widehat{G}_1, \ldots, \widehat{G}_{q+1}\}$ and for each possible

tuple $(L, R, \mathcal{B}, \widehat{C}, \widehat{p}) \in \Lambda_i$, we solve the corresponding CMWC instance $\mathcal{I} = (\widehat{G}_i - (L \cup R), (Q_{i-1} \cup Q_i) \setminus (L \cup R), \mathcal{B}, \mathcal{P}|_{V(G_i-(L\cup R))}, \widehat{p})$. That is, we set $\Lambda_i(L, R, \mathcal{B}, \widehat{C}, \widehat{p})$ if $\mathcal{I}$ is a yes-instance (Lines 3 to 8). Having computed all those values, we then proceed to filling table Γ_1. Since G_0 is a subgraph of G_1, and $G_1 = \widehat{G}_1$, we simply set $\Gamma_1(X, \mathcal{A}, \overline{C}, \overline{p}) = \Lambda_1(\emptyset, X, \mathcal{A}, \overline{C}, \overline{p})$ (for all tuples). This is justified by the fact that a solution is not allowed to delete any vertex in Q_0. To complete table Γ_i, $i > 1$, we simply use the following:

$$\Gamma_i(X, \mathcal{A}, \overline{C}, \overline{p}) = \bigvee [\Gamma_{i-1}(X', \mathcal{A}', \overline{C}', \overline{p}') \wedge \Lambda_i(L, R, \mathcal{B}, \widehat{C}, \widehat{p})],$$

where tuples $(X, \mathcal{A}, \overline{C}, \overline{p})$, $(X', \mathcal{A}', \overline{C}', \overline{p}')$, and $(L, R, \mathcal{B}, \widehat{C}, \widehat{p})$ are compatible. Finally, **ALG2** returns true whenever there exists a tuple $\Gamma_{q+1}(\emptyset, \mathcal{T}, C, p) = \mathsf{true}$ (for some $p \leq k$).

4.2 Correctness and runtime analysis

We are now ready to prove our main structural lemma which reduces the computation of the entries in Γ_i (when $i > 1$) to those in Γ_{i-1} and Λ_i. The lemma is proved in a purely existential setting and serves as the proof of correctness of the algorithm.

▶ **Lemma 4.7.** *For any $i \in [q+1]$ and tuple $\tau_1 = (X, \mathcal{A}, C_1, p_1) \in \Gamma_i$, $\mathbb{I}_{\tau_1}$ is a yes-instance if and only if there is a tuple $\tau_2 = (L, R, \mathcal{B}, C_2, p_2) \in \Lambda_i$ and a tuple $\tau_3 = (X', \mathcal{A}', C_3, p_3) \in \Gamma_{i-1}$ such that $\mathbb{I}_{\tau_2}$ and $\mathbb{I}_{\tau_3}$ are both yes-instances and all three tuples are compatible.*

▶ **Theorem 4.8.** Colorful Multiway Cut *can be solved in $\mathcal{O}^\star((k+t)^{\mathcal{O}(kt+k^3)} 2^{\mathcal{O}(\ell k)})$ time, where $t = |T|$.*

▶ **Corollary 4.9.** Simultaneous FVS/OCT *can be solved in $\mathcal{O}^\star(k^{\mathsf{poly}(\alpha,k)})$ time.*

Proof. Recall that Lemmas 3.1 and 3.2 together imply that if we can solve an instance of Colorful Separator in $\mathcal{O}^\star(k^{\mathsf{poly}(\alpha,k)})$ time then the algorithm for Simultaneous FVS/OCT follows. Any instance of Colorful Separator can be reduced to an instance of Colorful Multiway Cut with $|T| = 2$. From Theorem 4.8, such an instance can be solved in time $\mathcal{O}^\star(k^{\mathcal{O}(k^3)} 2^{\mathcal{O}(\alpha k)})$. ◀

5 W[1]-hardness of Simultaneous OCT

In this section we show that Simultaneous OCT is W[1]-hard. For notational convenience, we shall use a different encoding of α-edge-colored graphs. Given a graph G with vertex set $V(G)$ and edge set $E(G)$, we define a coloring function $\mathsf{col}(e) \subseteq 2^{[\alpha]}$. In particular, when $\alpha = 2$, we have $\mathsf{col}(e) \subseteq 2^{\{1,2\}}$. We start by establishing W[1]-hardness of Simultaneous Cut, which is formally defined below.

Simultaneous Cut	**Parameter(s):** k and α
Input: A graph G, two vertices $s, t \in V(G)$, an integer k, and a coloring function $\mathsf{col} : E(G) \to 2^{[\alpha]}$.	
Question: Is there $X \subseteq V(G) \setminus \{s, t\}$ of size at most k such that, for all $i \in [\alpha]$, $G_i - X$ has no (s,t)-paths? Here, for $i \in [\alpha]$, $G_i = (V(G), E_i)$, where $E_i = \{e \in E(G) \mid i \in \mathsf{col}(e)\}$.	

We give a parameterized reduction from Multicolored Clique which is known to be W[1]-hard [7]. The Multicolored Clique problem is formally defined below.

Multicolored Clique	**Parameter(s):** k
Input: A k-partite graph G with a partition $V_1, V_2, \ldots, V_k$ of $V(G)$ such that for all $i, j \in [k]$, $\lvert V_i \rvert = \lvert V_j \rvert$.	
Question: Is there $X \subseteq V(G)$ such that, for all $i \in [k]$, $\lvert X \cap V_i \rvert = 1$ and $G[X]$ is a clique?	

Given an instance $(G, V_1, V_2, \ldots, V_k)$ of Multicolored Clique, we proceed by creating an instance $(G', s, t, k', \mathsf{col}' : E(G') \to 2^{\{1,2\}})$ of Simultaneous Cut such that $(G, V_1, V_2, \ldots, V_k)$ is a yes-instance of Multicolored Clique if and only if $(G', s, t, k', \mathsf{col}' : E(G') \to 2^{\{1,2\}})$ is a yes-instance of Simultaneous Cut.

The intuitive description of the parameterized reduction is as follows. Let $(G, V_1, V_2, \ldots, V_k)$ be an instance of Multicolored Clique. Since $\lvert V_i \rvert = \lvert V_j \rvert$, for all $i, j \in [k]$, we assume that $\lvert V_i \rvert = \lvert V_j \rvert = n$. Furthermore, we assume that for every $i, j \in [k]$, $i \neq j$, there is at least one edge between V_i and V_j, otherwise, the instance is a trivial no-instance of Multicolored Clique and our reduction will simply output a trivial no-instance of Simultaneous Cut with $\alpha = 2$. For each $i \in [k]$ we assume an arbitrary (but fixed) ordering on the vertices in V_i. For each $i \in [k]$, we will have a vertex selection gadget $\mathcal{S}_i$ that will be responsible for selecting a vertex in V_i. To achieve this, $\mathcal{S}_i$ will have $k-1$ copies of each vertex in V_i, so that each vertex in V_i has a copy corresponding to every $j \in [k] \setminus \{i\}$. For each $j \in [k] \setminus \{i\}$, we have an (s,t)-path with all edges having color 1. Each path contains exactly one copy of every vertex in V_i. Furthermore, these vertices appear in the order given by the ordering we already fixed on the vertices of V_i.

The jth copy of the vertex set V_i will be used to ensure that there is an edge between the selected vertex in V_i and a vertex in V_j. The copies of any single vertex will form an (s,t)-separator of size $k-1$. Furthermore, the size of minimum (s,t)-separator in $\mathcal{S}_i$ will be $k-1$ and there will be exactly n distinct minimum separator each of which will correspond to a set comprising of $k-1$ copies of a vertex in V_i. By construction of the gadget and by setting budget constraints appropriately we will ensure that we must select a vertex from each of the $k-1$ copies of V_i, for each $i \in [k]$ and the selected $k-1$ vertices correspond to copies of the same vertex, i.e. we select a minimum separator. This will ensure that we have selected exactly one vertex from each V_i, for $i \in [k]$.

For $i, j \in [k]$, $i \neq j$, we will have edge selection gadgets E_{ij} which will ensure that there is an edge selected between V_i and V_j, and the selected edge is incident to the vertex selected from the vertex selection gadget. Finally, we will have a compatibility gadget which will ensure that the edges selected by E_{ij} and E_{ji} correspond to the same edge in G. We need to differentiate between gadgets E_{ij} and E_{ji} for technical reasons that will become clear later. We will now move to the formal description of the reduction.

Construction. Initially, $V(G') = \emptyset$ and $E(G') = \emptyset$. We add two special vertices s and t to $V(G')$, which are the vertices we want to separate, and which will be common to all the gadgets. For $i \in [k]$ we let v^i_j be the jth vertex in V_i. We now formally describe the construction of the various gadgets. We note that the gadgets are not necessarily vertex or edge disjoint (in addition to intersecting with $\{s, t\}$).

Vertex Selection Gadget. For each $i \in [k]$ we have a vertex selection gadget $\mathcal{S}_i$ defined as follows. For each $j \in [k] \setminus \{i\}$, $\mathcal{S}_i$ contains vertices in $V_{ij} = \{v^i_{j1}, v^i_{j2}, \ldots, v^i_{jn}\}$. Here, the vertices $v^i_{j1}, v^i_{j2}, \ldots, v^i_{jn}$ corresponds to one copy of the vertices $v^i_1, v^i_2, \ldots v^i_n$ in V_i. Note that for $j, j' \in [k] \setminus \{i\}$ vertices $v^i_{j\ell}, v^i_{j'\ell}$ correspond to copies of the same vertex, namely $v^i_\ell \in V_i$. For $i \in [k]$ and $\ell \in [n]$, we let $V^i_\ell = \{v^i_{j\ell} \mid j \in [k] \setminus \{i\}\}$, i.e. V^i_ℓ denotes the set comprising of

$k-1$ copies of the vertex $v_\ell^i \in V_i$. For $i \in [k]$, $\ell \in [n-1]$, and for each $u \in V_\ell^i$ and $u' \in V_{\ell+1}^i$ we add the edge $(u,u') \in E(G')$ and set $\mathsf{col}'((u,u')) = \{1\}$. Note that $G'[V_\ell^i \cup V_{\ell+1}^i]$ is a complete bipartite graph with all edges having the color 1 in their color set. For $i \in [k]$, $u \in V_1^i$ we add the edge $(s,u) \in E(G')$ and set $\mathsf{col}'((s,u)) = \{1\}$. Similarly, for $i \in [k]$, $u \in V_n^i$ we add the edge $(u,t) \in E(G')$ and set $\mathsf{col}'((u,t)) = \{1\}$.

Edge Selection Gadget. For $i \in [k]$ and $j \in [k] \setminus \{i\}$ the edge selection gadget $\mathcal{E}_{ij}$ is constructed as follows. The vertex set of $\mathcal{E}_{ij}$ contains a vertex $e_{uu'}$, for each edge $(u,u') \in E(G)$ with $u \in V_i$ and $u' \in V_j$. We note here that $\mathcal{E}_{ij}$ and $\mathcal{E}_{ji}$ denote distinct gadgets. For $\ell \in [n]$, we let $E_\ell^{ij} = \{e_{v_\ell^i u'} \mid u' \in V_j, (v_\ell^i, u) \in E(G)\}$, i.e. E_ℓ^{ij} contains vertices corresponding to those edges between V_i and V_j that are incident to the vertex $v_\ell^i \in V_i$. We let $E_{ij} = \cup_{\ell \in [n]} E_\ell^{ij}$. For $\ell \in [n]$ and each $u \in E_\ell^{ij}$, we add the edge $(u, v_{j\ell}^i)$ to $\mathcal{E}_{ij}$. We add an induced path P_ℓ^{ij} on the vertices in E_ℓ^{ij} (where the vertices appear in the natural order implied by the ordering of the vertices in V_j) and add these edges to $\mathcal{E}_\ell^{ij}$. For each edge $e \in E(P_\ell^{ij})$, we let $\mathsf{col}'(e) = \{2\}$. For $\ell \in [n+1]$, we let $\mathbf{K}_\ell^{ij}$ denote a $K_{3,3}$ (complete bipartite graph with 3 vertices on both side) with vertex bipartition $(\{p_\ell^{ij}, q_\ell^{ij}, r_\ell^{ij}\}, \{\bar{p}_\ell^{ij}, \bar{q}_\ell^{ij}, \bar{r}_\ell^{ij}\})$ and add it to $\mathcal{E}_{ij}$. We will refer to $\mathbf{K}_\ell^{ij}$s as *barrier blocks* of $\mathcal{E}_{ij}$. Finally, we join s, t and E_ℓ^{ij}, for $\ell \in [n]$ using the barrier blocks. This is done as follows.

For $\ell \in [n]$, let a_ℓ^{ij}, b_ℓ^{ij} be the first and the last vertex respectively, in the path P_ℓ^{ij}. We add the edges $(a_\ell^{ij}, \bar{p}_\ell^{ij}), (a_\ell^{ij}, \bar{q}_\ell^{ij}), (a_\ell^{ij}, \bar{r}_\ell^{ij})$ and $(b_\ell^{ij}, p_{\ell+1}^{ij}), (b_\ell^{ij}, q_{\ell+1}^{ij}), (b_\ell^{ij}, r_{\ell+1}^{ij})$ to $E(\mathcal{E}_{ij})$. Also, for $\ell \in [n]$, we add the edges $(v_{j\ell}^i, \bar{p}_\ell^{ij}), (v_{j\ell}^i, \bar{q}_\ell^{ij}), (v_{j\ell}^i, \bar{r}_\ell^{ij})$ and $(v_{j\ell}^i, p_{\ell+1}^{ij}), (v_{j\ell}^i, q_{\ell+1}^{ij}), (v_{j\ell}^i, r_{\ell+1}^{ij})$ to $E(\mathcal{E}_{ij})$. In addition, we add the edges $(s, p_1^{ij}), (s, q_1^{ij}), (s, \bar{r}_1^{ij})$, $(\bar{p}_{n+1}^{ij}, t), (\bar{q}_{n+1}^{ij}, t), (\bar{r}_{n+1}^{ij}, t)$ to $\mathcal{E}_{ij}$. For each $e \in E(\mathcal{E}_{ij})$, we set $\mathsf{col}'(e) = \{2\}$. This completes the description of the edge selection gadget.

Edge Compatibility Gadget. This gadget is used to ensure that the edge selected by $\mathcal{E}_{ij}$ and $\mathcal{E}_{ji}$ corresponds to the same edge of G. For $i,j \in [k]$, $i < j$, the edge compatibility gadget $\mathcal{C}_{ij}$ is constructed as described below. Basically, $\mathcal{C}_{ij}$ comprises of a set of edges between vertices in E_{ij} and vertices in E_{ji}. Recall that E_{ij} and E_{ji} contains vertices corresponding to the same edges, namely the edges between V_i and V_j in G. Hence, we can think of E_{ji} as a set comprising of a copy of the vertices in E_{ij}. We fix a lexicographic ordering on vertices in E_{ij} which we obtain as follows. For $e_{v_a^i, v_x^j}, e_{v_b^i, v_y^j} \in E_{ij}$, $e_{v_a^i, v_x^j} < e_{v_b^i, v_y^j}$ if (i) $a < b$ or (ii) $a = b$ and $x < y$. We denote the ordering of vertices in E_{ij} by $e_1^{ij}, e_2^{ij}, \ldots, e_m^{ij}$. Note this also fixes an ordering of vertices in E_{ji} which we denote by $e_1^{ji}, e_2^{ji}, \ldots, e_m^{ji}$. Here, m is the number of edges between V_i and V_j in G. For $\ell \in [m-1]$, we add the edges $(e_\ell^{ij}, e_{\ell+1}^{ij}), (e_\ell^{ij}, e_{\ell+1}^{ji}), (e_\ell^{ji}, e_{\ell+1}^{ij}), (e_\ell^{ji}, e_{\ell+1}^{ji})$ to $\mathcal{C}_{ij}$. That is we add all the edges in the bipartition between each consecutive pair of vertices in the ordered sets E_{ij} and E_{ji}. We add edges $(s, e_1^{ij}), (s, e_1^{ji})(e_m^{ij}, t), (e_m^{ji}, t)$ to $\mathcal{C}_{ij}$. For each edge $e \in \mathcal{C}_{ij}$, we set $\mathsf{col}'(e) = \{1\}$. We note here that in case we have created multiple edges say e, e' between vertices u, v then we delete e' and set $\mathsf{col}'(e) := \mathsf{col}'(e) \cup \mathsf{col}'(e')$.

We finally set $k' = k(k-1) + 2\binom{k}{2}$. We denote the graph constructed above by G' with the coloring function on the edge set denoted by col'.

▶ **Lemma 5.1.** *$(G, V_1, V_2, \ldots, V_k)$ is a yes-instance of* Multicolored Clique *if and only if $(G', s, t, k', \mathsf{col}' : E(G') \to 2^{\{1,2\}})$ is a yes-instance of* Simultaneous OCT.

▶ **Theorem 5.2.** *For all $\alpha \geq 2$,* Simultaneous Cut *is* W[1]-hard *when parameterized by k. Here, α is the number of colors in the coloring function of the edge set.*

We give a parameterized reduction from Simultaneous Cut to Simultaneous OCT, which implies the following theorem.

▶ **Theorem 5.3.** *For all $\alpha \geq 2$,* Simultaneous OCT *is* W[1]-hard *when parameterized by k. Here, α is the number of colors in the coloring function of the edge set.*

6 Conclusion

In light of Theorem 4.8, it is natural to ask whether one can improve the running time of our algorithm for Colorful Multiway Cut. In particular, is it possible to solve the problem in $\mathcal{O}^\star(k^{\mathcal{O}(k)})$ time when the number of terminals is constant and the number of colors is at most k? Another interesting question which remains open is whether the Simultaneous FVS/OCT problem admits a (randomized) polynomial kernel. Finally, we would also like to point out another interesting consequence of Theorem 5.3, i.e. the fact that Simultaneous OCT is W[1]-hard when parameterized by k. If we replace minimal feedback vertex sets by minimal odd cycle transversals in Lemma 2.3 then Theorem 5.3 implies that such a lemma cannot be true.

Acknowledgements. The authors are thankful to Saket Saurabh for helpful discussions.

References

1. Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous feedback vertex set: A parameterized perspective. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 7:1–7:15, 2016.
2. Leizhen Cai and Junjie Ye. Dual connectedness of edge-bicolored graphs and beyond. In *39th International Symposium on Mathematical Foundations of Computer Science*, pages 141–152, 2014.
3. Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21:1–21:35, 2015.
4. Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.
5. Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved Algorithms for Feedback Vertex Set Problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.
6. Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved Upper Bounds for Vertex Cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.
7. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
8. Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset Feedback Vertex Set Is Fixed-Parameter Tractable. *SIAM Journal of Discrete Mathematics*, 27(1):290–309, 2013.
9. Reinhard Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 4th edition, 2010.
10. Rod G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1997.
11. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, (FOCS)*, pages 470–479, 2012.
12. Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Transactions on Algorithms*, 13(2):29:1–29:32, 2017.
13. Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

14 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1749–1761, 2014.

15 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1802–1811, 2014.

16 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016.

17 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Information Processing Letters*, 114(10):556–560, 2014.

18 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):15:1–15:31, 2014.

19 Daniel Lokshtanov and M. S. Ramanujan. Parameterized Tractability of Multiway Cut with Parity Constraints. In *39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 750–761, 2012.

20 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. A linear time parameterized algorithm for directed feedback vertex set. *ArXiv e-prints*, 2016. `arXiv:1609.04347`.

21 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30:1–30:35, 2013.

22 M. S. Ramanujan and Saket Saurabh. Linear time parameterized algorithms via skew-symmetric multicuts. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1739–1748, 2014.

23 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

24 Junjie Ye. A note on finding dual feedback vertex set. *ArXiv e-prints*, 2015. `arXiv:1510.00773`.

A Composition Theorem for Randomized Query Complexity*

Anurag Anshu[1], Dmitry Gavinsky[†2], Rahul Jain[3], Srijita Kundu[4], Troy Lee[5], Priyanka Mukhopadhyay[6], Miklos Santha[‡7], and Swagato Sanyal[8]

1 **Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543**
a0109169@u.nus.edu
2 **Institute of Mathematics, Czech Academy of Sciences, 115 67 Žitna 25, Praha 1, Czech Republic**
3 **Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543 and MajuLab, UMI 3654, Singapore**
rahul@comp.nus.edu.sg
4 **Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543 and MajuLab, UMI 3654, Singapore**
srijita.kundu@u.nus.edu
5 **SPMS, Nanyang Technological University, 21 Nanyang Link, Singapore 637371 and Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543**
troyjlee@gmail.com
6 **Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543 and MajuLab, UMI 3654, Singapore**
a0109168@u.nus.edu
7 **IRIF, Université Paris Diderot, CNRS, 75205 Paris, France and Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543**
santha@irif.fr
8 **SPMS, Nanyang Technological University, 21 Nanyang Link, Singapore 637371 and Centre for Quantum Technologies, National University of Singapore, Block S15, 3 Science Drive 2, Singapore 117543**
ssanyal@ntu.edu.sg

Abstract

Let the randomized query complexity of a relation for error probability ϵ be denoted by $\mathsf{R}_\epsilon(\cdot)$. We prove that for any relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ and Boolean function $g : \{0,1\}^m \to \{0,1\}$, $\mathsf{R}_{1/3}(f \circ g^n) = \Omega(\mathsf{R}_{4/9}(f) \cdot \mathsf{R}_{1/2-1/n^4}(g))$, where $f \circ g^n$ is the relation obtained by composing f and g. We also show using an XOR lemma that $\mathsf{R}_{1/3}\left(f \circ \left(g^{\oplus}_{O(\log n)}\right)^n\right) = \Omega(\log n \cdot \mathsf{R}_{4/9}(f) \cdot \mathsf{R}_{1/3}(g))$, where $g^{\oplus}_{O(\log n)}$ is the function obtained by composing the XOR function on $O(\log n)$ bits and g.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes

* This work was partially supported by the National Research Foundation, including under NRF RF Award No. NRF-NRFF2013-13, the Prime Minister's Office, Singapore and the Ministry of Education, Singapore under the Research Centres of Excellence programme and by Grant No. MOE2012-T3-1- 009.
† D.G. is partially funded by the grant P202/12/G061 of GA ČR and by RVO: 67985840.
‡ M. S. is partially funded by the ANR Blanc program under contract ANR-12-BS02-005 (RDAM project).

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 10; pp. 10:1–10:13

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

Keywords and phrases Query algorithms and complexity, Decision trees, Composition theorem, XOR lemma, Hardness amplification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.10

1 Introduction

Given two Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, the composed function $f \circ g^n : (\{0,1\}^m)^n \to \{0,1\}$ is defined as follows: For $x = (x^{(1)}, \ldots, x^{(n)}) \in (\{0,1\}^m)^n$, $f \circ g^n(x) = f(g(x^{(1)}), \ldots, g(x^{(n)}))$. Composition of Boolean functions has long been a topic of active research in complexity theory. In many works, composition of Boolean function is studied in the context of a certain complexity measure. The objective is to understand the relation between the complexity of the composed function in terms of the complexities of the individual functions. Let $\mathsf{D}(\cdot)$ denote the deterministic query complexity. It is easy to see that $\mathsf{D}(f \circ g^n) \leq \mathsf{D}(f) \cdot \mathsf{D}(g)$ since $f \circ g$ can be computed by simulating an optimal query algorithm of f; whenever the algorithm makes a query, we simulate an optimal query algorithm of g and serve the query. It can be shown by an adversary argument that this is an optimal query algorithm and $\mathsf{D}(f \circ g^n) = \mathsf{D}(f) \cdot \mathsf{D}(g)$ [11, 15].

However, such a characterization is not so obvious for randomized query complexity. Although a similar upper bound still holds true (possibly accommodating a logarithmic overhead), it is no more as clear that it also asymptotically bounds the randomized query complexity of $f \circ g^n$ from below. Let $\mathsf{R}_\epsilon(\cdot)$ denote the ϵ-error randomized query complexity. Our main theorem in this work is the following.

▶ **Theorem 1** (Main Theorem). *For any relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ and Boolean function $g : \{0,1\}^m \to \{0,1\}$,*

$$\mathsf{R}_{1/3}(f \circ g^n) = \Omega(\mathsf{R}_{4/9}(f) \cdot \mathsf{R}_{1/2-1/n^4}(g)).$$

See Section 2 for definitions of composition and various complexity measures of relations. Theorem 1 implies that if g is a function that is hard to compute with error $1/2 - 1/n^4$, $f \circ g^n$ is hard to compute with error $1/3$.

In the special case where f is a function, Theorem 1 implies that $\mathsf{R}_{1/3}(f \circ g^n) = \Omega(\mathsf{R}_{1/3}(f) \cdot \mathsf{R}_{1/2-1/n^4}(g))$, since the success probability of query algorithms for functions can be boosted from $5/9$ to $2/3$ by constantly many independent repetitions followed by taking a majority of the different outputs.

Theorem 1 is useful only when the function g is hard against randomized query algorithms even for error $1/2 - 1/n^4$. In Section 3.1 we prove the following consequence of Theorem 1.

▶ **Theorem 2.** *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be any relation. Let $g : \{0,1\}^m \to \{0,1\}$ be a function. Let $g_t^{\oplus} : (\{0,1\}^m)^t \to \{0,1\}$ be defined as follows: for $x = (x^{(1)}, \ldots, x^{(t)}) \in (\{0,1\}^m)^t$, $g_t^{\oplus}(x) = \oplus_{i=1}^t g(x^{(i)})$. Then,*

$$\mathsf{R}_{1/3}\left(f \circ \left(g^{\oplus}_{O(\log n)}\right)^n\right) = \Omega(\log n \cdot \mathsf{R}_{4/9}(f) \cdot \mathsf{R}_{1/3}(g)).$$

Theorem 2 is proved by establishing, via an XOR lemma by Andrew Drucker [5], that if g is hard for error $1/3$ then $g^{\oplus}_{O(\log n)}$ is hard for error $1/2 - 1/n^4$.

Composition theorems for randomized query complexity have been extensively studied in the past. Göös and Jayram [6] showed a composition theorem for a constrained version of conical junta degree, which is a lower bound on randomized query complexity. Composition theorem for approximate degree (which also bounds randomized query complexity from below) for the special case of TRIBES function has seen a long line of research culminating in independent works of Sherstov [14] and Bun and Thaler [3] who settle the question by proving optimal bounds.

Composition theorem has been studied and shown in the context of communication and query complexities by the works of Göös, Pitassi and Watson [7, 8], Chattopadhyay et al. [4] when the function g is the indexing function or the inner product function with large enough arity. The work of Hatami, Hosseini and Lovett [9] proves a composition theorem in the context of communication and parity query complexites when the function g is the two-bit XOR function. Ben-David and Kothari [1] proved a composition theorem for the *sabotage complexity* of Boolean functions, a novel complexity measure defined in the same work that the authors prove to give quadratically tight bound on the randomized query complexity.

Composition theorems have also been successfully used in the past in constructing separating examples for various complexity measures, and bounding one complexity measure in terms of another. Kulkarni and Tal [10] proved an upper bound on fractional block sensitivity in terms of degree by analyzing the behavior of fractional block sensitivity under function composition. Separation between block sensitivity and degree was obtained by composing Kushilevitz's *hemi-icosahedron function* repeatedly with itself [13, 12, 2]. Separation between parity decision tree complexity and Fourier sparsity has been obtained by O'Donnell et al. by studying the behavior of *parity kill number* under function composition [13].

1.1 Our techniques

In this section, we give a high level overview of our proof of Theorem 1. We refer the reader to Section 2 for formal definitions of composition and various complexity measures of relations.

Let $\epsilon = 1/2 - 1/n^4$. Let μ be the distribution over the domain $\{0,1\}^m$ of g for which $\mathsf{R}_\epsilon(g)$ is achieved, i.e., $\mathsf{R}_\epsilon(g) = \mathsf{D}^\mu_\epsilon(g)$ (see Fact 3). For $b \in \{0,1\}$, let μ_b denote the distribution obtained by conditioning μ to the event that $g(x) = b$ (see Section 2 for a formal definition).

We show that for every probability distribution λ over the domain $\{0,1\}^n$ of f, there exists a deterministic query algorithm $\mathcal{A}$ with worst case query complexity at most $\mathsf{R}_{1/3}(f \circ g^n)/\mathsf{R}_\epsilon(g)$, such that $\Pr_{z\sim\lambda}[(z, \mathcal{A}(z)) \in f] \geq 5/9$. By the minimax principle (Fact 3) this proves Theorem 1.

Now using the distribution λ over $\{0,1\}^n$ we define a probability distribution γ over $(\{0,1\}^m)^n$. To define γ, we begin by defining a family of distributions $\{\gamma^z : z \in \{0,1\}^n\}$ over $(\{0,1\}^m)^n$. For a fixed $z = (z_1, \ldots, z_n) \in \{0,1\}^n$, we define γ^z by giving a sampling procedure:

1. For each $i = 1, \ldots, n$, sample $x^{(i)} = (x^{(i)}_1, \ldots, x^{(i)}_m)$ from $\{0,1\}^m$ independently according to μ_{z_i}.
2. Return $x = (x^{(1)}, \ldots, x^{(n)})$.

Thus for $z = (z_1, \ldots, z_n) \in \{0,1\}^n$ and $x = (x^{(1)}, \ldots, x^{(n)}) \in (\{0,1\}^m)^n$, $\gamma^z(x) = \Pi^n_{i=1} \mu_{z_i}(x^{(i)})$. Note that γ^z is supported only on strings x for which the following is true: for each $r \in \mathcal{R}$, $(x, r) \in f \circ g^n$ if and only if $(z, r) \in f$.

Having defined the distributions γ^z, we define the distribution γ by giving a sampling procedure:

1. Sample a $z = (z_1, \ldots, z_n)$ from $\{0,1\}^n$ according to λ.
2. Sample an $x = (x^{(1)}, \ldots, x^{(n)})$ from $(\{0,1\}^m)^n$ according to γ^z. Return x.

By minimax principle (Fact 3), there is a deterministic query algorithm $\mathcal{B}$ of worst case complexity at most $\mathsf{R}_{1/3}(f \circ g^n)$ such that $\Pr_{x\sim\gamma}[(x, \mathcal{B}(x)) \in f \circ g^n] \geq 2/3$. We will use $\mathcal{B}$ to construct a randomized query algorithm $\mathcal{A}'$ for f with the desired properties. A deterministic query algorithm $\mathcal{A}$ for f with required performance guarantees can then be obtained by appropriately fixing the randomness of $\mathcal{A}'$.

See Algorithm 1 for a formal description of $\mathcal{A}'$. Given an input $z = (z_1, \ldots, z_n)$, $\mathcal{A}'$ simulates $\mathcal{B}$. Recall that an input to $\mathcal{B}$ is an nm bit long string $(x_j^{(i)})_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}}$. Whenever $\mathcal{B}$ asks for (queries) an input bit $x_j^{(i)}$, a response bit is appropriately generated and passed to $\mathcal{B}$. To generate a response to a query by $\mathcal{B}$, a bit in z may be queried; those queries will contribute to the query complexity of $\mathcal{A}'$. The queries are addressed as follows. Let the simulation of $\mathcal{B}$ request bit $x_j^{(i)}$.

- If less than $\mathsf{D}^{\mu}_{\epsilon}(g)$ queries have been made into $x^{(i)}$ (including the current query) then a bit b is sampled from the marginal distribution of $x_j^{(i)}$ according to μ, conditioned on the responses to the past queries. b is passed to the simulation of $\mathcal{B}$.
- If $\mathsf{D}^{\mu}_{\epsilon}(g)$ queries have been made into $x^{(i)}$ (including the current query) then first the input bit z_i is queried; then a bit b is sampled from the marginal distribution of $x_j^{(i)}$ according to μ_{z_i}, conditioned on the responses to the past queries. b is passed to the simulation of $\mathcal{B}$.

The simulation of $\mathcal{B}$ continues until $\mathcal{B}$ terminates in a leaf. Then $\mathcal{A}'$ also terminates and outputs the label of the leaf.

We use Claims 8 and 9 to prove that for a fixed $z \in \{0,1\}^n$, the probability distribution induced by $\mathcal{A}'$ on the leaves of $\mathcal{B}$ is statistically close to the probability distribution induced by $\mathcal{B}$ on its leaves for a random input from γ^z. Averaging over different z's, the correctness of $\mathcal{A}'$ follows from the correctness of $\mathcal{B}$. The reader is referred to Section 3 for the details.

2 Preliminaries

In this section, we define some basic concepts, and set up our notations. We begin with defining the 2-sided error randomized and distributional query complexity measures of relations. The relations considered in this work will all be between the Boolean hypercube $\{0,1\}^k$ of some dimension k, and an arbitrary set $\mathcal{S}$. The strings $x \in \{0,1\}^n$ will be called as inputs to the relation, and $\{0,1\}^n$ will be referred to as the *input space* and the *domain* of relations.

▶ **Definition 1** (2-sided Error Randomized Query Complexity)**.** Let $\mathcal{S}$ be any set. Let $h \subseteq \{0,1\}^k \times \mathcal{S}$ be any relation and $\epsilon \in [0, 1/2)$. The 2-sided error randomized query complexity $\mathsf{R}_{\epsilon}(h)$ is the minimum number of queries made in the worst case by a randomized query algorithm $\mathcal{A}$ (the worst case is over inputs and the internal randomness of $\mathcal{A}$) that on each input $x \in \{0,1\}^k$ satisfies $\Pr[(x, \mathcal{A}(x)) \in h] \geq 1 - \epsilon$ (where the probability is over the internal randomness of $\mathcal{A}$).

▶ **Definition 2** (Distributional Query Complexity). Let $h \subseteq \{0,1\}^k \times \mathcal{S}$ be any relation, μ a distribution on the input space $\{0,1\}^k$ of h, and $\epsilon \in [0, 1/2)$. The distributional query complexity $\mathsf{D}^{\mu}_{\epsilon}(h)$ is the minimum number of queries made in the worst case (over inputs) by a deterministic query algorithm $\mathcal{A}$ for which $\Pr_{x\sim\mu}[(x, \mathcal{A}(x)) \in h] \geq 1 - \epsilon$.

In particular, if h is a function and $\mathcal{A}$ is a randomized or distributional query algorithm computing h with error ϵ, then $\Pr[h(x) = \mathcal{A}(x)] \geq 1 - \epsilon$, where the probability is over the respective sources of randomness.

The following theorem is von Neumann's minimax principle stated for decision trees.

▶ **Fact 3** (minimax principle). *For any integer k, set $\mathcal{S}$, and relation $h \subseteq \{0,1\}^k \times \mathcal{S}$,*

$$\mathsf{R}_{\epsilon}(h) = \max_{\mu} \mathsf{D}^{\mu}_{\epsilon}(h).$$

Let $g : \{0,1\}^m \to \{0,1\}$ be a Boolean function. Let μ be a probability distribution on $\{0,1\}^m$ which intersects non-trivially both with $g^{-1}(0)$ and with $g^{-1}(1)$. For each $z \in \{0,1\}$, let μ_z be the distribution obtained by restricting μ to $g^{-1}(z)$. Formally,

$$\mu_z(x) = \begin{cases} 0 & \text{if } g(x) \neq z \\ \frac{\mu(x)}{\sum_{y:g(y)=z} \mu(y)} & \text{if } g(x) = z \end{cases}$$

Notice that μ_0 and μ_1 are defined with respect to some Boolean function g, which will always be clear from the context.

▶ **Definition 4** (Subcube, Co-dimension). A subset $\mathcal{C}$ of $\{0,1\}^m$ is called a subcube if there exists a set $S \subseteq \{1, \ldots, m\}$ of indices and an *assignment function* $A : S \to \{0,1\}$ such that $\mathcal{C} = \{x \in \{0,1\}^m : \forall i \in S, x_i = A(i)\}$. The co-dimension $\mathsf{codim}(\mathcal{C})$ of $\mathcal{C}$ is defined to be $|S|$.

Let $\mathcal{C} \subseteq \{0,1\}^m$ be a subcube and μ be a probability distribution on $\{0,1\}^m$. We will often abuse notation and use $\mathcal{C}$ to denote the event that a random string x belongs to the subcube $\mathcal{C}$. The probability $\Pr_{x\sim\mu}[x \in \mathcal{C}]$ will be denoted by $\Pr_{\mu}[\mathcal{C}]$. For subcubes $\mathcal{C}_1$ and $\mathcal{C}_2$, the conditional probability $\Pr_{x\sim\mu}[x \in \mathcal{C}_2 \mid x \in \mathcal{C}_1]$ will be denoted by $\Pr_{\mu}[\mathcal{C}_2 \mid \mathcal{C}_1]$.

▶ **Definition 5** (Bias of a subcube). Let $g : \{0,1\}^m \to \{0,1\}$ be a Boolean function. Let μ be a probability distribution over $\{0,1\}^m$. Let $\mathcal{C} \subseteq \{0,1\}^m$ be a subcube such that $\Pr_{\mu}[\mathcal{C}] > 0$. The bias of $\mathcal{C}$ with respect to μ, $\mathsf{bias}^{\mu}(\mathcal{C})$, is defined to be:

$$\mathsf{bias}^{\mu}(\mathcal{C}) = |\Pr_{x\sim\mu}[g(x) = 0 \mid x \in \mathcal{C}] - \Pr_{x\sim\mu}[g(x) = 1 \mid x \in \mathcal{C}]|.$$

A Boolean function g is implicit in the definition of bias, which will always be clear from the context.

▶ **Proposition 6.** Let $g : \{0,1\}^m \to \{0,1\}$ be a Boolean function, and $\mathsf{D}^{\mu}_{\epsilon}(g) > 0$. Then,

$$\min_{b\in\{0,1\}} \{\Pr_{x\sim\mu}[g(x) = b]\} > \epsilon.$$

In particular, $\mathsf{bias}^{\mu}(\{0,1\}^m) < 1 - 2\epsilon$.

Proof. Towards a contradiction, assume that $\min_{b\in\{0,1\}}\{\Pr_{x\sim\mu}[g(x) = b]\} \leq \epsilon$. Then, the algorithm that outputs $\arg\max_{b\in\{0,1\}}\{\Pr_{x\sim\mu}[g(x) = b]\}$ makes 0 query and is correct with probability at least $1 - \epsilon$. This contradicts the hypothesis that $\mathsf{D}^{\mu}_{\epsilon}(g) > 0$. ◀

Now we define composition of two relations.

▶ **Definition 7** (Composition of relations)**.** Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ and $g \subseteq \{0,1\}^m \times \{0,1\}$ be two relations. The composed relation $f \circ g^n \subseteq (\{0,1\}^m)^n \times \mathcal{R}$ is defined as follows: For $x = (x^{(1)}, \ldots, x^{(n)}) \in (\{0,1\}^m)^n$ and $r \in \mathcal{R}$, $(x, r) \in f \circ g^n$ if and only if there exists $b = (b^{(1)}, \ldots, b^{(n)}) \in \{0,1\}^n$ such that for each $i = 1, \ldots, n$, $(x^{(i)}, b^{(i)}) \in g$ and $(b, r) \in f$.

We will often view a deterministic query algorithm as a binary decision tree. In each vertex v of the tree, an input variable is queried. Depending on the outcome of the query, the computation goes to a child of v. The child of v corresponding to outcome b to the query made is denoted by v_b. It is well known that the set of inputs that lead the computation of a decision tree to a certain vertex forms a subcube. We will denote the subcube corresponding to a vertex v by $\mathcal{C}_v$.

We next prove two claims about bias, probability and co-dimension of subcubes that will be useful. Claim 8 states that for a function with large distributional query complexity, the bias of most shallow leaves of any deterministic query procedure is small.

▶ **Claim 8.** Let $g : \{0,1\}^m \to \{0,1\}$ be a Boolean function. Let $\epsilon \in [1/4, 1/2)$ and let $\delta = 1/2 - \epsilon$. Let μ be a probability distribution on $\{0,1\}^m$, and $\mathsf{D}^{\mu}_{\epsilon}(g) = c > 0$. Let $\mathcal{B}$ be any deterministic query algorithm for strings in $\{0,1\}^m$. For each $y \in \{0,1\}^m$, let ℓ_y be the unique leaf of $\mathcal{B}$ that contains y. Then,

(a) $\Pr_{y\sim\mu}[\mathsf{codim}(\ell_y) < c \text{ and } \mathsf{bias}^{\mu}(\ell_y) \geq 2\delta^{1/2}] < \delta^{1/2}$.
(b) For each $b \in \{0,1\}$, $\Pr_{y\sim\mu_b}[\mathsf{codim}(\ell_y) < c \text{ and } \mathsf{bias}^{\mu}(\ell_y) \geq 2\delta^{1/2}] < 4\delta^{1/2}$.

In the above claim $\mathcal{B}$ could just be a deterministic procedure that makes queries and eventually terminates; whether or not it makes any output upon termination is not of any consequence here.

Proof. We first show that part (a) implies part (b). To this end, assume part (a) and fix a $b \in \{0,1\}$. Let $a(y)$ be the indicator variable for the event $\mathsf{codim}(\ell_y) < c$ and $\mathsf{bias}^{\mu}(\ell_y) \geq 2\delta^{1/2}$. Thus, part (a) states that $\Pr_{y\sim\mu}[a(y) = 1] < \delta^{1/2}$. Now,

$$\begin{aligned}
&\Pr_{y\sim\mu_b}[\mathsf{codim}(\ell_y) < c \text{ and } \mathsf{bias}^{\mu}(\ell_y) \geq 2\delta^{1/2}] \\
&= \sum_{y:a(y)=1} \mu_b(y) \\
&= \frac{1}{\sum_{y:g(y)=b} \mu(y)} \sum_{y:a(y)=1} \mu(y) \quad \text{(From the definition of } \mu_b\text{)} \\
&< \frac{1}{\epsilon} \Pr_{y\sim\mu}[a(y) = 1] \quad \text{(From Proposition 6)} \\
&< 4\delta^{1/2}. \quad \text{(By the hypothesis } \epsilon \geq 1/4 \text{ and part (a))}
\end{aligned}$$

We now prove part (a). Towards a contradiction assume that

$$\Pr_{y\sim\mu}[\mathsf{codim}(\ell_y) < c \text{ and } \mathsf{bias}^{\mu}(\ell_y) \geq 2\delta^{1/2}] \geq \delta^{1/2}.$$

Now consider the following query algorithm $\mathcal{A}$ on m bit strings:

Begin simulating $\mathcal{B}$. Let $\mathcal{C}$ be the subcube associated with the current node of $\mathcal{B}$ in the simulation. Simulate $\mathcal{B}$ unless one of the following happens.

- $\mathcal{B}$ terminates.
- The number of queries made is $c-1$.
- $\mathsf{bias}^{\mu}(\mathcal{C}) \geq 2\delta^{1/2}$.

Upon termination, if $\mathsf{bias}^{\mu}(\mathcal{C}) \geq 2\delta^{1/2}$, output $\arg\max_{b\in\{0,1\}} \Pr_{y\sim\mu}[g(y) = b \mid y \in \mathcal{C}]$. Else output a uniformly random bit.

It immediately follows that the worst case query complexity of $\mathcal{A}$ is at most $c-1$. Now, we will prove that $\Pr_{y\sim\mu}[\mathcal{A}(y) = g(y)] \geq 1-\epsilon$. This will contradict the hypothesis that $\mathsf{D}^{\mu}_{\epsilon}(g) = c$. Let $\mathcal{L}$ be the node of $\mathcal{B}$ at which the computation of $\mathcal{A}$ ends. Let $\Pr_{y\sim\mu}[\mathsf{bias}^{\mu}(\mathcal{L}) \geq 2\delta^{1/2}] = p$. By our assumption, the probability (over μ) that $\mathcal{L}$ is a leaf and $\mathsf{bias}^{\mu}(\mathcal{L}) \geq 2\delta^{1/2}$ is at least $\delta^{1/2}$; in particular $p \geq \delta^{1/2}$. Now,

$$
\begin{aligned}
&\Pr_{y\sim\mu}[\mathcal{A}(y) = g(y)] \\
&= \Pr_{y\sim\mu}[\mathsf{bias}^{\mu}(\mathcal{L}) \geq 2\delta^{1/2}] \cdot \Pr_{y\sim\mu}[\mathcal{A}(y) = g(y) \mid \mathsf{bias}^{\mu}(\mathcal{L}) \geq 2\delta^{1/2}] + \\
&\qquad \Pr_{y\sim\mu}[\mathsf{bias}^{\mu}(\mathcal{L}) < 2\delta^{1/2}] \cdot \Pr_{y\sim\mu}[\mathcal{A}(y) = g(y) \mid \mathsf{bias}^{\mu}(\mathcal{L}) < 2\delta^{1/2}] \\
&\geq p \cdot (1/2 + \delta^{1/2}) + (1-p).\frac{1}{2} \quad \text{(from our assumption)} \\
&= 1/2 + p \cdot \delta^{1/2} \\
&\geq 1/2 + \delta \quad (\text{since } p \geq \delta^{1/2}) \\
&= 1 - \epsilon.
\end{aligned}
$$

This completes the proof. ◀

The next claim states that if a subcube has low bias with respect to a distribution μ, then the distributions μ_0 and μ_1 ascribe almost the same probability to it.

▶ **Claim 9.** Let $g : \{0,1\}^m \to \{0,1\}$ be a Boolean function and $\delta \in (0, \frac{1}{2}]$. Let μ be a distribution on $\{0,1\}^m$. Let $\mathcal{C}$ be a subcube such that $\Pr_{\mu}[\mathcal{C}] > 0$ and $\mathsf{bias}^{\mu}(\mathcal{C}) \leq \delta$. Also assume that $\mathsf{bias}^{\mu}(\{0,1\}^m) \leq \delta$. Then for any $b \in \{0,1\}$ we have,

(a) $\Pr_{\mu}[\mathcal{C}] \leq (1+4\delta) \cdot \Pr_{\mu_b}[\mathcal{C}]$,
(b) $\Pr_{\mu}[\mathcal{C}] \geq (1-4\delta) \cdot \Pr_{\mu_b}[\mathcal{C}]$.

Proof. We prove part (a) of the claim. The proof of part (b) is similar.

By the definition of bias and the hypothesis, for each $b \in \{0,1\}$,

$$\sum_{y\in\mathcal{H}_m : g(y)=b} \mu(y) \leq \left(\frac{1}{2} + \frac{\delta}{2}\right) \cdot \sum_{y\in\mathcal{H}_m} \mu(y) = \frac{1}{2} + \frac{\delta}{2}, \tag{1}$$

$$\sum_{y\in\mathcal{C} : g(y)=b} \mu(y) \geq \left(\frac{1}{2} - \frac{\delta}{2}\right) \cdot \sum_{y\in\mathcal{C}} \mu(y) > 0. \tag{2}$$

Now,

$$
\begin{aligned}
&\Pr_{\mu_b}[\mathcal{C}] = \sum_{y\in\mathcal{C}} \mu_b(y) \\
&= \frac{\sum_{y\in\mathcal{C} : g(y)=b} \mu(y)}{\sum_{y\in\mathcal{H}_m : g(y)=b} \mu(y)}
\end{aligned}
$$

$$\geq \frac{(1/2-\delta/2)\cdot\sum_{y\in\mathcal{C}}\mu(y)}{1/2+\delta/2} \quad \text{(From Equations (1) and (2)}$$
$$= \frac{1/2-\delta/2}{1/2+\delta/2}\cdot\Pr_{\mu}[\mathcal{C}]$$

Thus,

$$\Pr_{\mu}[\mathcal{C}] \leq \frac{1/2+\delta/2}{1/2-\delta/2}\cdot\Pr_{\mu_b}[\mathcal{C}] \leq (1+4\delta)\cdot\Pr_{\mu_b}[\mathcal{C}]. \quad \text{(since } \delta\leq\tfrac{1}{2}\text{)}$$ ◀

3 Composition Theorem

In this section we prove our main theorem. We restate it below.

▶ **Theorem 1** (Main Theorem). *For any relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ and Boolean function $g : \{0,1\}^m \to \{0,1\}$,*

$$\mathsf{R}_{1/3}(f\circ g^n) = \Omega(\mathsf{R}_{4/9}(f)\cdot\mathsf{R}_{1/2-1/n^4}(g)).$$

Proof. We begin by recalling the notations defined in Section 1.1 that we will use in this proof.

Let $\epsilon = 1/2 - 1/n^4$. Let μ be the distribution over the domain $\{0,1\}^m$ of g for which $\mathsf{R}_\epsilon(g)$ is achieved, i.e., $\mathsf{R}_\epsilon(g) = \mathsf{D}^\mu_\epsilon(g)$. (see Fact 3)

We show that for every probability distribution λ over the input space $\{0,1\}^n$ of f, there exists a deterministic query algorithm $\mathcal{A}$ with worst case query complexity at most $\mathsf{R}_{1/3}(f\circ g)/\mathsf{R}_\epsilon(g)$, such that $\Pr_{z\sim\lambda}[(z,\mathcal{A}(z))\in f] \geq 5/9$. By the minimax principle (Fact 3) this will prove Theorem 1.

Using λ, we define a probability distribution γ over $(\{0,1\}^m)^n$. We first define a family of distributions $\{\gamma^z : z\in\{0,1\}^n\}$ over $(\{0,1\}^m)^n$. For a fixed $z\in\{0,1\}^n$, we define γ^z by giving a sampling procedure:

1. For each $i = 1,\ldots,n$, sample $x^{(i)} = (x^{(i)}_1,\ldots,x^{(i)}_m)$ from $\{0,1\}^m$ independently according to μ_{z_i}.
2. Return $x = (x^{(1)},\ldots,x^{(n)})$.

Thus for $z = (z_1,\ldots,z_n)\in\{0,1\}^n$ and $x = (x^{(1)},\ldots,x^{(n)})\in(\{0,1\}^m)^n$, $\gamma^z(x) = \Pi_{i=1}^n \mu_{z_i}(x^{(i)})$. Note that γ^z is supported only on strings x for which the following is true: for each $r\in\mathcal{R}$, $(x,r)\in f\circ g^n$ if and only if $(z,r)\in f$.

Now, we define the distribution γ by giving a sampling procedure:

1. Sample a $z = (z_1,\ldots,z_n)$ from $\{0,1\}^n$ according to λ.
2. Sample an $x = (x^{(1)},\ldots,x^{(n)})$ from $(\{0,1\}^m)^n$ according to γ^z. Return x.

By the minimax principle (Fact 3), there is a deterministic query algorithm $\mathcal{B}$ of worst case complexity at most $\mathsf{R}_{1/3}(f\circ g^n)$ such that $\Pr_{x\sim\gamma}[(x,\mathcal{B}(x))\in f\circ g^n]\geq 2/3$. We will use $\mathcal{B}$ to construct a randomized query algorithm $\mathcal{A}'$ for f with the desired properties. A deterministic query algorithm $\mathcal{A}$ for f with required performance guarantees can then be obtained by appropriately fixing the randomness of $\mathcal{A}'$. Algorithm 1 formally defines the algorithm $\mathcal{A}'$ that we construct.

Algorithm 1: Randomized query algorithm $\mathcal{A}'$ for f

Input: $z \in \{0,1\}^n$

1 Initialize $v \leftarrow$ root of the decision tree $\mathcal{B}$, $Q \leftarrow \emptyset$
2 **while** v *is not a leaf* **do**
3 Let a bit in $x^{(i)}$ be queried at v
4 **if** $i \notin Q$ **then** `/* codim(`$\mathcal{C}_v^{(i)}$`) < `$\mathsf{D}_\epsilon^\mu(g)$ `if this is satisfied */`
5 Set $v \leftarrow v_b$ with probability $\Pr_\mu[\mathcal{C}_{v_b}^{(i)} \mid \mathcal{C}_v^{(i)}]$
6 **if** $\mathsf{codim}(\mathcal{C}_v^{(i)}) = \mathsf{D}_\epsilon^\mu(g) - 1$ **then**
7 Query z_i
8 Set $Q \leftarrow Q \cup \{i\}$
9 **else**
10 **if** $\Pr_{\mu_{z_i}}[\mathcal{C}_v^{(i)}] = 0$ **then**
11 Output 0
12 **else**
13 Set $v \leftarrow v_b$ with probability $\Pr_{\mu_{z_i}}[\mathcal{C}_{v_b}^{(i)} \mid \mathcal{C}_v^{(i)}]$
14 Output label of v

From the definition of bias one can verify that the event in step 5 in Algorithm 1 that is being conditioned on, has non-zero probabilities under the respective distribution; hence, the probabilistic process is well-defined.

From the description of $\mathcal{A}'$ it is immediate that z_i is queried only if the underlying simulation of $\mathcal{B}$ queries at least $\mathsf{R}_\epsilon(g)$ locations in $x^{(i)}$. Thus the worst-case query complexity of $\mathcal{A}'$ is at most $\mathsf{R}_{1/3}(f \circ g^n)/\mathsf{R}_\epsilon(g)$.

We are left with the task of bounding the error of $\mathcal{A}'$. Let $\mathcal{L}$ be the set of leaves of the decision tree $\mathcal{B}$. Each leaf $\ell \in \mathcal{L}$ is labelled with $b_\ell \in \mathcal{R}$; whenever the computation reaches ℓ, b_ℓ is output.

For a vertex v, let the corresponding subcube $\mathcal{C}_v$ be $\mathcal{C}_v^{(1)} \times \ldots \times \mathcal{C}_v^{(n)}$, where $\mathcal{C}_v^{(i)}$ is a subcube of the domain of the i-th copy of g (corresponding to the input $x^{(i)}$). Recall from Section 2 that for $b \in \{0,1\}$, v_b denotes the b-th child of v.

For each leaf $\ell \in \mathcal{L}$ and $i = 1, \ldots, n$, define $\mathsf{snip}^{(i)}(\ell)$ to be 1 if there is a node t in the unique path from the root of $\mathcal{B}$ to ℓ such that $\mathsf{codim}(\mathcal{C}_t^{(i)}) < \mathsf{D}_\epsilon^\mu(g)$ and $\mathsf{bias}^\mu(\mathcal{C}_t^{(i)}) \geq \frac{2}{n^2}$, and 0 otherwise. Define $\mathsf{snip}(\ell) := \vee_{i=1}^n \mathsf{snip}^{(i)}(\ell)$.

For each $\ell \in \mathcal{L}$, define p_ℓ^z to be the probability that for an input drawn from γ^z, the computation of $\mathcal{B}$ terminates at leaf ℓ. We have,

$$\Pr_{x \sim \gamma^z}[(x, \mathcal{B}(x)) \in f \circ g^n] = \Pr_{x \sim \gamma^z}[(z, \mathcal{B}(x)) \in f] = \sum_{\ell \in \mathcal{L} : (z, b_\ell) \in f} p_\ell^z. \tag{3}$$

From our assumption about $\mathcal{B}$ we also have that,

$$\Pr_{x \sim \gamma}[(x, \mathcal{B}(x)) \in f \circ g^n] = \mathop{\mathbb{E}}_{z \sim \lambda} \Pr_{x \sim \gamma^z}[(x, \mathcal{B}(x)) \in f \circ g^n] \geq \frac{2}{3}. \tag{4}$$

Now, consider a run of $\mathcal{A}'$ on z. For each $\ell \in \mathcal{L}$ of $\mathcal{B}$, define q_ℓ^z to be the probability that the computation of $\mathcal{A}'$ on z terminates at leaf ℓ of $\mathcal{B}$. Note that the probability is over the internal randomness of $\mathcal{A}'$.

To finish the proof, we need the following two claims. The first one states that the leaves $\ell \in \mathcal{L}$ are sampled with similar probabilities by $\mathcal{B}$ and $\mathcal{A}'$.

▶ **Claim 10.** For each $\ell \in \mathcal{L}$ such that $\mathsf{snip}(\ell) = 0$, and for each $z \in \{0,1\}^n$, $\frac{8}{9} \cdot p_\ell^z \leq q_\ell^z \leq \frac{10}{9} \cdot p_\ell^z$.

The next Claim states that for each z, the probability according to γ^z of the leaves ℓ for which $\mathsf{snip}(\ell) = 1$ is small.

▶ **Claim 11.**

$$\forall z \in \{0,1\}^n, \sum_{\ell \in \mathcal{L}, \mathsf{snip}(\ell)=1} p_\ell^z \leq \frac{4}{n}.$$

We first finish the proof of Theorem 1 assuming Claims 10 and 11, and then prove the claims. For a fixed input $z \in \{0,1\}^n$, the probability that $\mathcal{A}'$, when run on z, outputs an r such that $(z,r) \in f$, is at least

$$\begin{aligned}
&\sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f, \mathsf{snip}(\ell)=0}} q_\ell^z \geq \sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f, \mathsf{snip}(\ell)=0}} \frac{8}{9} \cdot p_\ell^z \quad \text{(By Claim 10)} \\
&= \frac{8}{9} \left(\sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f}} p_\ell^z - \sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f, \mathsf{snip}(\ell)=1}} p_\ell^z \right) \\
&\geq \frac{8}{9} \left(\sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f}} p_\ell^z - \frac{4}{n} \right). \quad \text{(By Claim 11)} \qquad (5)
\end{aligned}$$

Thus, the success probability of $\mathcal{A}'$ is at least

$$\begin{aligned}
\mathop{\mathbb{E}}_{z \sim \lambda} \sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f, \mathsf{snip}(\ell)=0}} q_\ell^z &\geq \frac{8}{9} \cdot \left(\mathop{\mathbb{E}}_{z \sim \lambda} \sum_{\substack{\ell \in \mathcal{L}, \\ (z,b_\ell) \in f}} p_\ell^z - \frac{4}{n} \right) \quad \text{(By Equation (5))} \\
&\geq \frac{8}{9} \cdot \left(\frac{2}{3} - \frac{4}{n} \right) \quad \text{(By Equations (3) and (4))} \\
&\geq \frac{5}{9}. \quad \text{(For large enough } n\text{)}
\end{aligned}$$

We now give the proofs of Claims 10 and 11.

Proof of Claim 10. We will prove the first inequality. The proof of the second inequality is similar[1].

Fix a $z \in \{0,1\}^n$ and a leaf $\ell \in \mathcal{L}$ such that $\mathsf{snip}(\ell) = 0$. For each $i = 1, \ldots, n$, assume that $\mathsf{codim}(\mathcal{C}_\ell^{(i)}) = d^{(i)}$, and in the path from the root of $\mathcal{B}$ to ℓ the variables $x_1^{(i)}, \ldots, x_{d^{(i)}}^{(i)}$ are set to bits $b_1, \ldots, b_{d^{(i)}}$ in this order.

We first observe that if v is a vertex of $\mathcal{B}$ for which the condition in step 10 of A' is satisfied, then co-dimension of $\mathcal{C}_v^{(i)}$ is at most $\mathsf{D}_\epsilon^\mu(g) - 1$, and $\mathsf{bias}(\mathcal{C}_v^{(i)}) = 1$; hence each leaf ℓ' of $\mathcal{B}$ reachable from v has $\mathsf{snip}(\ell') = 1$.

[1] Note that only the first inequality is used in the proof of Theorem 1.

The computation of $\mathcal{A}'$ terminates at leaf ℓ if the values of the different bits $x_j^{(i)}$ sampled by $\mathcal{A}'$ agree with the leaf ℓ. The probability of that happening is given by

$$q_\ell^z = \prod_{i=1}^{n} \Pr_{\mathcal{A}'}[x_1^{(i)} = b_1, \ldots, x_{d^{(i)}}^{(i)} = b_{d^{(i)}} \mid z] \tag{6}$$

$$= \prod_{i=1}^{n} \Pr_{x\sim\mu}[x_1^{(i)} = b_1, \ldots, x_{\mathsf{D}_\epsilon^\mu(g)-1}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)-1}] \cdot \Pr_{x\sim\mu_{z_i}}[x_{\mathsf{D}_\epsilon^\mu(g)}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)}, \ldots, x_{d^{(i)}}^{(i)} = b_{d^{(i)}} \mid x_1^{(i)} = b_1, \ldots, x_{\mathsf{D}_\epsilon^\mu(g)-1}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)-1}]. \tag{7}$$

The second equality above follows from the observation that in Algorithm 1, the first $\mathsf{D}_\epsilon^\mu(g)-1$ bits of $x^{(i)}$ are sampled from their marginal distributions with respect to μ, and the subsequent bits are sampled from their marginal distributions with respect to μ_{z_i}. In equation (7), the term $\Pr_{x\sim\mu_{z_i}}[x_{\mathsf{D}_\epsilon^\mu(g)}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)}, \ldots, x_{d^{(i)}}^{(i)} = b_{d^{(i)}} \mid x_1^{(i)} = b_1, \ldots, x_{\mathsf{D}_\epsilon^\mu(g)-1}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)-1}]$ is interpreted as 1 if $d^{(i)} < \mathsf{D}_\epsilon^\mu(g)$.

We invoke Claim 9(b) with $\mathcal{C}$ set to the subcube $\{x \in \{0,1\}^m : x_1^{(i)} = b_1, \ldots, x_{\mathsf{D}_\epsilon^\mu(g)-1}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)-1}\}$ and δ set to $\frac{2}{n^2}$. To see that the claim is applicable here, note that from the assumption $\mathsf{snip}(\ell) = 0$ we have that $\mathsf{bias}(\mathcal{C}) < \delta = \frac{2}{n^2} < \frac{1}{2}$, where the last inequality holds for large enough n. Also, since $\mathsf{D}_\epsilon^\mu(g) > 0$, by Proposition 6 the bias of $\{0,1\}^m$ is at most $\frac{2}{n^4} < \frac{2}{n^2} = \delta$. Continuing from Equation (7), by invoking Claim 9(b) we have,

$$q_\ell^z \geq \prod_{i=1}^{n}(1 - 8/n^2) \Pr_{x\sim\mu_{z_i}}[x_1^{(i)} = b_1, \ldots, x_{\mathsf{D}_\epsilon^\mu(g)-1}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)-1}] \cdot \Pr_{x\sim\mu_{z_i}}[x_{\mathsf{D}_\epsilon^\mu(g)}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)}, \ldots, x_{d^{(i)}}^{(i)} = b_{d^{(i)}} \mid x_1^{(i)} = b_1, \ldots, x_{\mathsf{D}_\epsilon^\mu(g)-1}^{(i)} = b_{\mathsf{D}_\epsilon^\mu(g)-1}]$$

$$= (1 - 8/n^2)^n \prod_{i=1}^{n} \Pr_{x\sim\mu_{z_i}}[x_1^{(i)} = b_1, \ldots, x_{d^{(i)}}^{(i)} = b_{d^{(i)}}]$$

$$\geq \frac{8}{9} \cdot p_\ell^z. \quad \text{(For large enough } n\text{)}$$

◀

Proof of Claim 11. Fix a $z \in \{0,1\}^n$. We shall prove that for each i, $\sum_{\ell\in\mathcal{L},\mathsf{snip}^{(i)}(\ell)=1} p_\ell^z \leq \frac{4}{n^2}$. That will prove the claim, since $\sum_{\ell\in\mathcal{L},\mathsf{snip}(\ell)=1} p_\ell^z \leq \sum_{i=1}^n \sum_{\ell\in\mathcal{L},\mathsf{snip}^{(i)}(\ell)=1} p_\ell^z$.

To this end, fix an $i \in \{1, \ldots, n\}$. For a random x drawn from γ^z, let p be the probability that in strictly less than $\mathsf{D}_\epsilon^\mu(g)$ queries the computation of $\mathcal{B}$ reaches a node t such that $\mathsf{bias}(\mathcal{C}_t^{(i)})$ is at least $\frac{2}{n^2}$. Note that this probability is over the choice of the different $x^{(j)}$'s. We shall show that $p \leq \frac{4}{n^2}$. This is equivalent to showing that $\sum_{\ell\in\mathcal{L},\mathsf{snip}^{(i)}(\ell)=1} p_\ell^z \leq \frac{4}{n^2}$.

Note that each $x^{(j)}$ is independently distributed according to μ_{z_j}. By averaging, there exists a choice of $x^{(j)}$ for each $j \neq i$ such that for a random $x^{(i)}$ chosen according to μ_{z_i}, a node t as above is reached within at most $\mathsf{D}_\epsilon^\mu(g) - 1$ steps with probability at least p. Fix such a setting for each $x^{(j)}$, $j \neq i$. Claim 11 follows from Claim 8 (note that $\epsilon = \frac{1}{2} - \frac{1}{n^4} \geq \frac{1}{4}$ for large enough n). ◀

This completes the proof of Theorem 1. ◀

3.1 Hardness Amplification Using XOR Lemma

In this section we prove Theorem 2.

Theorem 1 is useful only when the function g is hard against randomized query algorithms even for error $1/2 - 1/n^4$. In this section we use an XOR lemma to show a procedure that, given any g that is hard against randomized query algorithms with error $1/3$, obtains another function on a slightly larger domain that is hard against randomized query algorithms with error $1/2 - 1/n^4$. This yields the proof of Theorem 2.

Let $g : \{0,1\}^m \to \{0,1\}$ be a function. Let $g_t^{\oplus} : (\{0,1\}^m)^t \to \{0,1\}$ be defined as follows. For $x = (x^{(1)}, \ldots, x^{(t)}) \in (\{0,1\}^m)^t$,

$$g_t^{\oplus}(x) = \oplus_{i=1}^{t} g(x^{(i)}).$$

The following theorem is obtained by specializing Theorem 3 of Andrew Drucker's paper [5] to this setting.

▶ **Theorem 12** (Drucker 2011 [5] Theorem 3)**.**

$$\mathsf{R}_{1/2-2^{-\Omega(t)}}(g_t^{\oplus}) = \Omega(t \cdot \mathsf{R}_{1/3}(g)).$$

Theorem 2 (restated below) follows by setting $t = \Theta(\log n)$ and combining Theorem 12 with Theorem 1.

▶ **Theorem 2.** *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be any relation. Let $g : \{0,1\}^m \to \{0,1\}$ be a function. Let $g_t^{\oplus} : (\{0,1\}^m)^t \to \{0,1\}$ be defined as follows: for $x = (x^{(1)}, \ldots, x^{(t)}) \in (\{0,1\}^m)^t$, $g_t^{\oplus}(x) = \oplus_{i=1}^{t} g(x^{(i)})$. Then,*

$$\mathsf{R}_{1/3}\left(f \circ \left(g_{O(\log n)}^{\oplus}\right)^n\right) = \Omega(\log n \cdot \mathsf{R}_{4/9}(f) \cdot \mathsf{R}_{1/3}(g)).$$

References

1. Shalev Ben-David and Robin Kothari. Randomized query complexity of sabotaged and composed functions. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 60:1–60:14, 2016.
2. Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.
3. Mark Bun and Justin Thaler. Dual lower bounds for approximate degree and Markov-Bernstein inequalities. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 303–314, 2013.
4. Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation theorems via pseudorandom properties. *CoRR*, abs/1704.06807, 2017.
5. Andrew Drucker. Improved direct product theorems for randomized query complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, June 8-10, 2011*, pages 1–11, 2011.
6. Mika Göös and T. S. Jayram. A composition theorem for conical juntas. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 5:1–5:16, 2016.
7. Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1077–1088, 2015.
8. Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. *CoRR*, abs/1703.07666, 2017.

9 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 282–288, 2016.

10 Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chicago J. Theor. Comput. Sci.*, 2016, 2016.

11 Ashley Montanaro. A composition theorem for decision tree complexity. *Chicago J. Theor. Comput. Sci.*, 2014, 2014.

12 Noam Nisan and Avi Wigderson. On rank vs. communication complexity. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 831–836, 1994.

13 Ryan O'Donnell, John Wright, Yu Zhao, Xiaorui Sun, and Li-Yang Tan. A composition theorem for parity kill number. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 144–154, 2014.

14 Alexander A. Sherstov. Approximating the AND-OR tree. *Theory of Computing*, 9:653–663, 2013.

15 Avishay Tal. Properties and applications of boolean function composition. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 441–454, 2013.

Verification of Asynchronous Programs with Nested Locks

Mohamed Faouzi Atig[1], Ahmed Bouajjani[2], K. Narayan Kumar*[3], and Prakash Saivasan[4]

1 **Uppsala University, Sweden**
mohamed_faouzi.atig@it.uu.se
2 **IRIF, Université Paris Diderot, France**
abou@irif.fr
3 **Chennai Mathematical Institute and UMI RELAX, Chennai, India**
kumar@cmi.ac.in
4 **TU Braunschweig, Germany**
p.saivasan@tu-bs.de

Abstract

In this paper, we consider asynchronous programs consisting of multiple recursive threads running in parallel. Each of the threads is equipped with a multi-set. The threads can create tasks and post them onto the multi-sets or read a task from their own. In addition, they can synchronise through a finite set of locks. In this paper, we show that the reachability problem for such class of asynchronous programs is undecidable even under the nested locking policy. We then show that the reachability problem becomes decidable (Exp-space-complete) when the locks are not allowed to be held across tasks. Finally, we show that the problem is NP-complete when in addition to previous restrictions, threads always read tasks from the same state.

1998 ACM Subject Classification D.2.4 Software, Program Verification

Keywords and phrases Asynchronous programs, Nested-locks, Reachability Problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.11

1 Introduction

Asynchronous programming is widely used in building efficient and responsive software. Jobs are decomposed in tasks that are delegated to different threads running in parallel. These threads share a global memory, and they also have their own local memory. In addition, each thread has an unbounded buffer where the tasks posted to the thread are stored. These tasks are handled by the thread in a serial manner, i.e., by running each task until completion before taking another one. Each task corresponds to the execution of a sequential program that can access both (thread-)local and global variables, call (potentially recursive) procedures, and create new tasks that are posted to designated threads.

The behaviours of asynchronous programs can be extremely intricate and unpredictable due to the dynamic creation of concurrent tasks. This makes the reasoning about asynchronous programs extremely hard.

Therefore, developing automated techniques for the verification of asynchronous programs is an important and challenging research topic. In the case of single-thread asynchronous

* Partially supported by Indo-French Project AVeCSO, Indo-Swedish DST-VR Project P-02/2014 and the Infosys Foundation

programs, it has been shown that the reachability problem is decidable and EXPSPACE-complete (assuming that the data domain is finite) [2,5]. However, this problem is undecidable in general, and this is the case even for two threads handling only one task each [11]. Therefore, decidable instances of this problem can be obtained only by considering either under-approximations for bug detection using, e.g., bounded analyses [1, 4, 10], or by considering over-approximations for establishing absence of bugs, using abstract analyses that focus on relevant aspects.

In the context of abstract analyses of concurrent programs, one useful approach is abstracting away the content of the shared variables carrying data while keeping precise the content of the control variables, such as *locks*, that are used for synchronisation. Indeed, concurrency bugs are in general due to a misuse of synchronisation allowing unexpected interleavings of concurrent actions. Therefore, assuming that locks are the only shared variables between threads retains the relevant information, without being too coarse, when reasoning about the existence of, e.g., data races and deadlocks. This approach has been introduced in [8], where it has been shown that, for multi-thread programs with locks (and no task creation), the reachability problem is undecidable in general, and that this problem becomes decidable when locks are used (i.e., acquired and released) in a *well-nested* manner. This problem has been investigated further for larger classes of programs by other authors, e.g., in [6,9]. In particular, it has been shown that for dynamic networks of pushdown systems (modelling concurrent programs with thread creation), which is a class of models with a decidable reachability problem [3] that is incomparable with asynchronous programs, the extension with well-nested locking preserves the decidability of the reachability problem [6]. The goal of this paper is to investigate the decidability and the complexity of the reachability problem for asynchronous programs with nested locking.

We first prove that, surprisingly, the reachability of asynchronous programs with nested locks is undecidable as soon as four threads are considered (two with unbounded call stacks, and two being finite-state). However, we provide a condition on the use of locks by threads across task handling phases that leads to decidability. In fact, we found that the source of undecidability (even under nested locking) is the transfer of locks between tasks executed successively on a single thread. Therefore, we require that (1) a thread should not hold any lock when it starts handling a task, and (2) all locks acquired during the execution of a task must be released before completion of the task, i.e., when the handling of a task is completed, the set of locks held by the thread must be again empty. Technically, we define a task-locking policy that requires that every thread should not hold any lock when its stack is empty. We prove that under this policy, the reachability problem of asynchronous programs with nested locks is in EXPSPACE. The proof is by a polynomial reduction to the case of single-thread asynchronous programs using a nontrivial serialisability argument. Importantly, despite the high worst case complexity, there are existing work for solving efficiently the reachability problem of single-threaded asynchronous programs in practice [7].

Moreover, we consider an interesting and practically relevant case for which we establish a better complexity. In fact, we consider that while tasks should be allowed to communicate and synchronise through the shared global memory, they should not communicate through the thread-local shared memory. Indeed, this would assume that the tasks rely on the order they are scheduled, which is in general not under the control of the programmer. So, it is quite natural to assume that before termination, a task can put the result of its computation in the shared memory, and it can also create a new task that will be its continuation (when it will be scheduled later), but that the thread does not transfer any information to the next task it executes. (Actually, asynchronous programs have typically User Interface threads

(UI's) that consist of loops receiving jobs (or reacting to events), and creating for them handlers running in parallel.) Technically, we consider that a thread always starts handling tasks in the same state. Interestingly, we prove that under this assumption the reachability problem becomes in NP. The proof is by a reduction to the reachability problem in the case where the number of task handling phases is polynomially bounded.

To summarise, we prove that by forbidding transfers of locks between tasks executing on a same thread, the reachability problem of multithreaded asynchronous programs with nested locks is EXPSPACE-complete. Furthermore, we prove that by forbidding transfers of local states between tasks executing on a same thread, the reachability problem becomes NP-complete. Our results open the door to the development of efficient and complete methods for verifying asynchronous programs against concurrency bugs.

2 Preliminaries

Let Σ be a finite alphabet. We use Σ^* and Σ^+ to denote the set of all finite words and non-empty finite words, respectively. We use ϵ to denote the empty word. We write Σ_ϵ to denote $\Sigma \cup \{\epsilon\}$. For $w = a_1 a_2 \dots a_n \in \Sigma^*$, we let $|w| = n$, $w[i] = a_i$ and $w[i,j]$ to denote the length of w, the i^{th} letter a_i and the subword $a_i \dots a_j$, respectively. Given a word $w \in \Sigma$ and $\Sigma' \subseteq \Sigma$, we let $w \downarrow_{\Sigma'}$ to denote the projection of w onto Σ' .

A multi-set over Σ is a function $M : \Sigma \mapsto \mathbb{N}$. We denote by $M[\Sigma]$ the set of all multi-sets over Σ and by $\emptyset$ the empty multi-set. Given two multi-sets M and M', we write $M' \leq M$ iff $M'(a) \leq M(a)$ for all $a \in \Sigma$. We denote by $M + M'$ the multi-set formed by $(M + M')(a) = M(a) + M'(a)$ for all $a \in \Sigma$. For $M \geq M'$, $M - M'$ is defined in a similar manner. For any word $w \in \Sigma^*$, we denote by $\lfloor w \rfloor$ the multi-set formed by counting the number of occurrences of each letter from Σ in w.

A *pushdown automaton (PDA)* is a tuple $\mathbf{P} = (Q, \Gamma, \Sigma, \delta, s_0, \alpha_0)$ where Q is the finite set of states, Γ is the finite stack alphabet, Σ is the finite input alphabet, $s_0 \in Q$ is the initial state, $\alpha_0 \in \Gamma$ is the initial stack symbol, and δ is the transition relation. We assume that Γ contains the special stack bottom symbol $\perp$. The transition set δ is a subset of $Q \times \Gamma \times \Sigma_\epsilon \times \Gamma^* \times Q$ with the restrictions that: (1) if $\tau = (q, \alpha, a, \beta, q') \in \delta$ then $|\beta| \leq 2$ and (2) $\beta \in \{b\perp \mid b \in \Gamma_\epsilon\}$ when $\alpha = \perp$. We use $\mathbf{Src}(\tau) = q$ (resp. $\mathbf{Dest}(\tau) = q'$) to refer to the head (resp. tail) state q (resp. q') of the transition τ, and $\lambda(\tau)$ to denote the label a.

A configuration of $\mathbf{P}$ is a pair (q, γ) with $q \in Q$ and $\gamma \in ((\Gamma \setminus \{\perp\})^* \cdot \{\perp\})$. Given a configuration $c = (q, \gamma)$, we use $\mathbf{Stt}(c)$ (resp. $\mathbf{Stk}(c)$) to refer to the state q (resp. the stack component γ). The initial configuration of $\mathbf{P}$ is defined by the pair $(s_0, \alpha_0\perp)$. The transition relation $\xrightarrow{\tau}_{\mathbf{P}}$, with $\tau \in \delta$, relating pairs of configurations, is defined as the smallest relation satisfying the following condition: $(q, \alpha\gamma) \xrightarrow{\tau}_{\mathbf{P}} (q', \beta\gamma)$ if τ is of the form $(q, \alpha, a, \beta, q')$. This transition corresponds to the pop of the symbol α and the push of the word β.

We often omit the reference to $\mathbf{P}$ and write $\xrightarrow{\tau}$ when $\mathbf{P}$ is clear from the context. Sometimes, we omit τ and simply write $\rightarrow$ when the reference to τ is not important. We write $(q, \gamma) \xrightarrow{\sigma} (q', \gamma')$ for $\sigma = \tau_1 \dots \tau_n \in \delta^*$ to mean that there is a sequence of transitions of the form $(q, \gamma) = (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \xrightarrow{\tau_2} \dots \xrightarrow{\tau_{n-1}} (q_{n-1}, \gamma_{n-1}) \xrightarrow{\tau_n} (q_n, \gamma_n) = (q', \gamma')$. Given two configurations c_1, c_2, we use $L(\mathbf{P}, c_1, c_2)$ to denote the set of words w such that there is $\tau_1, \dots, \tau_n \in \delta$ with $c_1 \xrightarrow{\tau_1\tau_2\cdots\tau_n} c_2$ and $w = \lambda(\tau_1)\lambda(\tau_2)\cdots\lambda(\tau_n)$. We use $L(\mathbf{P}, c_2)$ to denote $L(\mathbf{P}, c_1, c_2)$ where $c_1 = (s_0, \alpha_0\perp)$ is the initial configuration.

3 Model

Multiset PushDown Systems (MPDS) have been introduced by Sen and Viswanathan as a formal model for asynchronous programs [12]. An MPDS model consists of a pushdown automaton equipped with a multiset. The multiset is used to store pending tasks (i.e., stack symbols). When the stack is empty, a pending task is taken, in non-deterministic manner, from the multiset and put into the stack. Then, the system starts the execution of the chosen task following the pushdown transition rules with the ability to create new tasks (that will be added to the multiset). In this paper, we consider a generalization of multi-set pushdown systems (MPDS) called N-Multi-set Pushdown Systems (N-MPDS) where $N \in \mathbb{N}$ denotes the (fixed) number of threads executing in parallel. An N-MPDS consists of a collection of pushdown automata (each one comes with its own multi-set). When the stack of a pushdown automaton is empty, a task is taken from its associated multi-set and executed. During the execution of a task, newly created tasks are added to a pre-determined multi-set. Furthermore, the pushdown automaton can communicate through a finite set of locks (i.e., each pushdown automaton can acquire and release a given particular lock). Thus, an MPDS (resp. pushdown automata communicating via locks [8]) corresponds to the particular case where we have one pushdown automaton (i.e., $N = 1$) (resp. there is no task creation).

▶ **Definition 1.** An N-MPDS over the (finite) set of locks $\mathcal{L}$ is a tuple $A = (\Sigma, \mathcal{P}, \mathcal{L})$, where Σ is a finite set of tasks and $\mathcal{P} = \{\mathbf{P}_i \mid 1 \leq i \leq N\}$ is a collection of pushdown automata (or threads) $\mathbf{P}_i = (Q_i, \Gamma_i, \mathcal{O}_i, \delta_i, s_i, \alpha_i)$, where $\mathcal{O}_i = \{i!j(a), i?a \mid a \in \Sigma, 1 \leq j \leq N\} \cup \{\mathrm{lck}_i(l), \mathrm{rel}_i(l) \mid l \in \mathcal{L}\}$. Here $i!j(a)$ means that the thread i creates a task labeled by a and adds it to the multi-set of thread j. While $i?a$ means that the thread i picks a pending task labeled by a from its multi-set. Furthermore, $\mathrm{lck}_i(l)/\mathrm{rel}_i(l)$ corresponds to acquiring / releasing of the lock l by the thread i. We assume that the sets δ_i, $1 \leq N$, are disjoint and let $\delta = \bigcup_{1 \leq i \leq N} \delta_i$. We also *require* that (1) $\delta_i \cap (Q_i \times (\Gamma_i \setminus \{\bot\}) \times \{i?a \mid a \in \Sigma\} \times \Gamma_i^* \times Q_i) = \emptyset$ for all $i \in \{1, \ldots, N\}$ (i.e., the execution of pending tasks can only be performed when the stack is empty), and (2) if a transition is of the form $(q, \bot, b, \beta\bot, q')$ is in δ, with $\beta \in \Sigma$, then b is of the form $i?\beta$ for some $i \in \{1, \ldots, N\}$ (i.e., the thread i picks a pending task β and adds it to its empty stack).

A configuration of an N-MPDS A is a triple of functions $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ where, for each $1 \leq i \leq N$, $\boldsymbol{c}(i)$ is a configuration of $\mathbf{P}_i$, $\boldsymbol{m}(i)$ is a multi-set over Σ (representing the set of tasks waiting to be executed by the thread i) and $\boldsymbol{l}(i) \subseteq \mathcal{L}$ is the set of locks held by thread i. We require that $\boldsymbol{l}(i) \cap \boldsymbol{l}(j) = \emptyset$ for all $i \neq j$. The initial configuration is defined $(\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0})$, where $\boldsymbol{c_0}(i) = (s_i, \bot), \boldsymbol{m_0}(i) = \lfloor \alpha_i \rfloor$ and $\boldsymbol{l_0}(i) = \emptyset$ for all $i, 1 \leq i \leq N$ (i.e., the stack of each thread is empty and there is only one pending task per thread and all locks are free). Observe that the definition of the initial configuration is equivalent to the one where each pushdown automaton is in its initial configuration, there is no pending task and all locks are free. We only use the former for the sake of simplicity.

Observe that an MPDS [12] can be defined as a 1-MPDS $A = (\Sigma, \mathcal{P}, \mathcal{L})$ whose set of locks is empty (i.e., $\mathcal{L} = \emptyset$) and set of threads $\mathcal{P}$ consists of only one pushdown automaton $\mathbf{P}_1 = (Q_1, \Gamma_1, \mathcal{O}_1, \delta_1, s_1, \alpha_1)$. To simplify the presentation, we use $(\Sigma, \mathbf{P}_1)$ to denote the MPDS A. Further, we use $(\boldsymbol{c}(1), \boldsymbol{m}(1))$ to denote a configuration of A instead of $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$.

Given two configurations $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ and $(\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$ and a transition $\tau \in \delta$, we use $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\tau}_A (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$ to denote that there is an index $i \in \{1, \ldots, N\}$ such that $\tau \in \delta_i$, $\boldsymbol{c}(i) \xrightarrow{\tau}_{\mathbf{P}_i} \boldsymbol{c'}(i)$, $\forall j \neq i$ we have $\boldsymbol{c}(j) = \boldsymbol{c'}(j)$ and further one of the following holds:

- $\lambda(\tau) = i!j(a)$: $\boldsymbol{m'}(j) = \boldsymbol{m}(j) + \lfloor a \rfloor$, $\boldsymbol{m}(k) = \boldsymbol{m'}(k)$ for all $k \neq j$ and $\boldsymbol{l} = \boldsymbol{l'}$. (The thread i creates a task of type a and adds it to the multiset of the thread j.)
- $\lambda(\tau) = i?a$: $\boldsymbol{m}(i)(a) > 1$, $\boldsymbol{m'}(i) = \boldsymbol{m}(i) - \lfloor a \rfloor$, $\boldsymbol{m}(j) = \boldsymbol{m'}(j)$ for all $k \neq i$, and $\boldsymbol{l} = \boldsymbol{l'}$.

(The thread i picks a task a from its multiset and adds it to its stack. Recall that removing a task from the multiset is possible only if the stack is empty.)

- $\lambda(\tau) = \text{lck}_i(l)$: $l \notin \bigcup_{1 \leq k \leq N} \boldsymbol{l}(k)$, $\boldsymbol{l'}(i) = \boldsymbol{l}(i) \cup \{l\}$, $\boldsymbol{m}(i) = \boldsymbol{m'}(i)$, $\boldsymbol{l}(k) = \boldsymbol{l'}(k)$ and $\boldsymbol{m}(k) = \boldsymbol{m'}(k)$ for all $k \neq i$. (The thread i acquires the lock l if it is not already held.)
- $\lambda(\tau) = \text{rel}_i(l)$: $l \in \boldsymbol{l}(i)$, $\boldsymbol{l'}(i) = \boldsymbol{l}(i) \setminus \{l\}$, $\boldsymbol{m}(i) = \boldsymbol{m'}(i)$, $\boldsymbol{l}(k) = \boldsymbol{l'}(k)$ and $\boldsymbol{m}(k) = \boldsymbol{m'}(k)$ for all $k \neq i$. (The thread i releases the lock l.)

An execution π of A is an alternating sequence $(\boldsymbol{c_1}, \boldsymbol{m_1}, \boldsymbol{l_1}) \cdot \tau_1 \cdot (\boldsymbol{c_2}, \boldsymbol{m_2}, \boldsymbol{l_2}) \cdot \tau_2 \cdots (\boldsymbol{c_{n-1}}, \boldsymbol{m_{n-1}}, \boldsymbol{l_{n-1}}) \cdot \tau_{n-1} \cdot (\boldsymbol{c_n}, \boldsymbol{m_n}, \boldsymbol{l_n})$ of configurations and transitions such that $(\boldsymbol{c_i}, \boldsymbol{m_i}, \boldsymbol{l_i}) \xrightarrow{\tau_i}_A (\boldsymbol{c_{i+1}}, \boldsymbol{m_{i+1}}, \boldsymbol{l_{i+1}})$, $\forall i \in \{1, \ldots, n-1\}$. For configurations $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ and $(\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$, we write $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\sigma}_A (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$ to denote that there is an execution $\pi = (\boldsymbol{c_1}, \boldsymbol{m_1}, \boldsymbol{l_1}) \cdot \tau_1 \cdot (\boldsymbol{c_2}, \boldsymbol{m_2}, \boldsymbol{l_2}) \cdot \tau_2 \cdots (\boldsymbol{c_{n-1}}, \boldsymbol{m_{n-1}}, \boldsymbol{l_{n-1}}) \cdot \tau_{n-1} \cdot (\boldsymbol{c_n}, \boldsymbol{m_n}, \boldsymbol{l_n})$ such that $\sigma = \tau_1 \tau_2 \cdots \tau_{n-1}$, $(\boldsymbol{c_1}, \boldsymbol{m_1}, \boldsymbol{l_1}) = (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ and $(\boldsymbol{c_n}, \boldsymbol{m_n}, \boldsymbol{l_n}) = (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$. Sometimes we write the execution π as $(\boldsymbol{c_1}, \boldsymbol{m_1}, \boldsymbol{l_1}) \xrightarrow{\tau_1}_A (\boldsymbol{c_2}, \boldsymbol{m_2}, \boldsymbol{l_2}) \ldots (\boldsymbol{c_{n-1}}, \boldsymbol{m_{n-1}}, \boldsymbol{l_{n-1}}) \xrightarrow{\tau_{n-1}}_A (\boldsymbol{c_n}, \boldsymbol{m_n}, \boldsymbol{l_n})$.

Reachability problem. Given a N-MPDS $A = (\Sigma, \mathcal{P}, \mathcal{L}, \Delta)$ (as defined above) and a function $\boldsymbol{r}$ (referred to as the destination) that assigns to each $i : 1 \leq i \leq N$ a state from $Q_i \setminus \{s_i\}$, the *reachability problem* asks if there is an execution of the form $(\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0}) \xrightarrow{\sigma}_A (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$, with $(\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0})$ is the initial configuration, for some $\boldsymbol{c}, \boldsymbol{m}$ and $\boldsymbol{l}$ such that for all $i : 1 \leq i \leq N$, we have $\textbf{Stt}(\boldsymbol{c}(i)) = \boldsymbol{r}(i)$.

First note that, without the assumption of removing a pending task from the multi-set when the stack of the thread is empty, we can simulate the intersection of two pushdown automata by a 2-MPDS, without any locks, by using the multi-sets as a synchronizing mechanism. Given two pushdown automata over an alphabet Σ, we construct a 2-MPDS with task alphabet $\Sigma \cup \{\#\}$. The simulation proceeds as follows: The first thread guesses a letter a, simulates a step of the first PDS, posts this letter a as a task to the second thread and waits for a task of type $\#$. The second thread nondeterministically guesses the next letter a, picks up a task of this type from its multi-set, simulates a step and then posts the task $\#$ to the first thread. Thus, we may simulate both the pushdowns on the same input word and the reachability problem for 2-MPDS without locks is rendered undecidable. Observe that, in this simulation, values are removed from the multi-sets at will and this goes against the spirit of our model: the multi-sets were introduced to hold the tasks that await execution. A thread should execute these (recursive) tasks one after another. In particular a task should be removed from the multi-set for execution only when the previous task has completed. When a task is completed, the call stack of the thread should be empty.

Second, the reachability problem for 2-MPDS without multi-sets is undecidable in the presence of locks. This follows from the undecidability of the reachability problem for pushdown automata synchronizing using locks [11]. Thus, we need restrictions on the usage of locks as well. One well-known restriction that yields decidability for networks of pushdown automata is that of *nested locking*. Nested locking, introduced by Kahlon et al. [8], requires that locks be released in the same order as they are acquired. This is formalized as follows.

Nested Locking. Given an execution of the form $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\tau_1}_A (\boldsymbol{c_1}, \boldsymbol{m_1}, \boldsymbol{l_1}) \xrightarrow{\tau_2}_A (\boldsymbol{c_2}, \boldsymbol{m_2}, \boldsymbol{l_2}) \ldots (\boldsymbol{c_{n-1}}, \boldsymbol{m_{n-1}}, \boldsymbol{l_{n-1}}) \xrightarrow{\tau_n}_A (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$, we say positions $i, j \in \{1, \ldots, n\}$ form a *acquire-release pair* if some lock l is acquired in the ith transition and released by the jth transition and this lock is not acquired or released in between i.e. there is a $k : 1 \leq k \leq N$, and $l \in \mathcal{L}$ such that $\lambda(\tau_i) = \text{lck}_k(l)$, $\lambda(\tau_j) = \text{rel}_k(l)$ and $\forall r \in \{i+1, \ldots, j-1\}, \lambda(\tau_r) \notin \bigcup_{1 \leq m \leq N} \{\text{lck}_m(l), \text{rel}_m(l)\}$. We write $i \curvearrowright_k j$ to indicate this. An execution of the form $\pi = (\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0}) \xrightarrow{\sigma} (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ is

said to be well-nested iff there are no positions i, j, i', j' such that $i < i' < j < j'$ and $i \curvearrowright_k j$ and $i' \curvearrowright_k j'$ for some $k, 1 \leq k \leq N$.

An N-MPDS A is said to be *well-nested* if all its executions of the form $\pi = (\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0}) \xrightarrow{\sigma} (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ are well-nested. Recall that $(\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0})$ is the initial configuration. As indicated earlier, Kahlon et al. [8] show that the reachability problem restricted to nested locking is decidable for networks of pushdown automata with locks. However, in the presence of multi-sets and locks, the nested locking assumption is still insufficient to obtain decidability.

4 Undecidability of the Reachability Problem for Well-Nested N-MPDSs

In this section we show that the reachability problem for N-MPDSs remains undecidable even when assuming the well nested locking policy. In particular we prove:

▶ **Theorem 2.** *The reachability problem for well-nested 4-MPDSs is undecidable.*

Proof sketch. Let P^1 and P^2 be two pushdown automata over an alphabet Σ. We construct a 4-MPDS A that simulates joint executions of P^1 and P^2. In the following, we assume w.l.o.g. that there are no ϵ moves in P^1 and P^2 (the undecidability result holds even with such an assumption). The MPDS A has four components P_1, P_2, P_3 and P_4 where P_1 and P_2 simulate P^1 and P^2 respectively, using the agents P_3 and P_4 to ensure that the simulations follow the same input word. In fact, P_3 and P_4 will not use their respective stacks.

The system A uses two locks l_1 and l_2 and the set of tasks is given by $\Sigma \cup \{a, b, r, l\}$. The simulation begins with an initialization step and this is followed by a sequence of steps, where in each step the threads P_1 and P_2 simulate a run of P^1 and P^2 on one letter.

In the initialization step, the thread P_3 acquires the lock l_1 and sends the task b to both P_1 and P_2 instructing them to begin the simulation. Both threads P_1 and P_2 await for the task b and begin their simulation on receiving this task. At the end of this initialization step (and at the beginning of each of the subsequent steps) the lock l_1 is with P_3 and l_2 is free and all the task multi-sets are empty.

In each step, the thread P_i, $1 \leq i \leq 2$, does the following: takes lock l_2, continues the simulation of P^i by reading a letter $c \in \Sigma$, posts the task c to the thread P_3, releases lock l_2 and then waits to take lock l_1. When available it takes l_1 and releases it immediately to complete its execution of the step.

In each step, the thread P_3 does the following: guesses a task type $c \in \Sigma$, removes two copies of c from its multi-set (ensuring that P_1 and P_2 have carried out simulations on the same letter), sends the task l to the thread P_4 (instructing it to take the lock l_2), waits for the task a (an acknowledgment from P_4 that it has indeed taken the lock l_2), releases the lock l_1 (to enable P_1 and P_2 to complete the concluding part of their execution of this step), retakes lock l_1, sends the task r to P_4 (instructing it to release the lock l_2) and waits for the task a (an acknowledgment from P_4 that it has indeed released the lock l_2).

In each step, the thread P_4 awaits the task l, then takes the lock l_2, sends the task a to P_3 in acknowledgment, awaits the task r, then releases lock l_2 and sends the task a to P_3.

It is clear that in each step, the simulation of both pushdowns is extended by a run on the same letter from Σ. We still have to argue that this protocol ensures that the threads proceed step by step (i.e., some of them cannot go ahead before the others are ready to participate in the next step). In each step, after simulating a run of the pushdowns both threads P_1 and P_2 have to wait for lock l_1 to be released. This is possible only after P_3 has verified that they have both used identical letters in their simulation. When the lock

l_1 is available for them to complete their executions of this step, the lock l_2 is guaranteed to be held by P_4 (since P_3 releases l_1 only after confirmation from P_4 that the lock l_2 has been taken). Thus after completing the current step P_1 and P_2 cannot proceed to the next step of the simulation (until l_2 is free). The thread P_3 takes back the lock l_1 before l_2 is released by P_4 and thus the locks are returned to the required state before the next step in the simulation begins.

It is possible that the lock l_1 is taken back by P_3 before P_1 or P_2 (or both) complete their final operations in the step. In this case, the system deadlocks since the thread that failed to complete will wait for l_1 while P_3 will wait for a task from Σ to be posted by this waiting thread. Thus the simulating threads can neither get ahead nor fall behind in each step. ◀

5 Well-Nested N-MPDS under the Task Locking Policy

As we have seen, allowing the transfer of locks, even in a nested manner, from a task to another task leads to the undecidability of the reachability problem for N-multi-set pushdown systems. Therefore, we consider an additional constraint on the locking policy. The new constraint consists in requiring that threads do not hold any locks when their stack is empty. This restriction can be understood as follows: the threads can be thought of as *schedulers* that pick and execute tasks. As tasks are executed to completion, a new task is picked only when the stack is empty, this amounts to requiring that it is mainly the tasks that use locks and not threads. However, since the thread and the tasks share the local state space, the thread is still allowed to pass a finite amount of local state information to the tasks, but it is not allowed to pass or receive locks. We shall return to this point in the next section.

Task Locking Policy. Let $A = (\Sigma, \mathcal{P}, \mathcal{L})$ be an N-MPDS. We say that an execution $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\tau_1}_A (\boldsymbol{c_1}, \boldsymbol{m_1}, \boldsymbol{l_1}) \xrightarrow{\tau_2}_A (\boldsymbol{c_2}, \boldsymbol{m_2}, \boldsymbol{l_2}) \ldots (\boldsymbol{c_{n-1}}, \boldsymbol{m_{n-1}}, \boldsymbol{l_{n-1}}) \xrightarrow{\tau_n}_A (\boldsymbol{c}_n, \boldsymbol{m}_n, \boldsymbol{l}_n)$ is a *task-locking* execution if for each $i : 1 \leq i \leq n$ and for each $j : 1 \leq j \leq N$, if $\mathbf{Stk}(\boldsymbol{c_i}(j)) = \perp$ then $\boldsymbol{l_i}(j) = \emptyset$. The N-MPDS A is under the *task-locking* policy if all its executions of the form $\pi = (\boldsymbol{c_0}, \boldsymbol{m_0}, \boldsymbol{l_0}) \xrightarrow{\sigma}_A (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l})$ are *task-locking* executions.

Our main result is that the reachability problem for well-nested N-MPDSs under the task-locking policy is decidable and is Expspace-Complete.

▶ **Theorem 3.** *The reachability problem for well-nested MPDSs under the task-locking policy is* Expspace-Complete.

The lower bound follows immediately from the Expspace-hardness of MPDS [12]. The rest of this section is dedicated to prove the upper-bound (namely that the reachability problem for well-nested N-MPDSs under the task-locking policy is in Expspace). As a first step, we show that one may *serialize* each execution of the system in such a way that (i) completed tasks can be executed atomically (ii) incomplete tasks (which are at most one per thread) can broken up in a bounded number of segments such that each segment can be executed atomically. This will be used in the next step to polynomially reduce our problem to the reachability in a (single) pushdown system with multi-sets.

For the rest of the section, let us fix an N-MPDS $A = (\Sigma, \mathcal{P}, \mathcal{L}, \Delta)$ where $\mathcal{P} = \{\mathbf{P}_i \mid 1 \leq i \leq N\}$ with $\mathbf{P}_i = (Q_i, \Gamma_i, \mathcal{O}_i, \delta_i, s_i, \alpha_i)$. We will further assume w.l.o.g. that in the N-MPDS A, every thread starts its execution by removing a task from the multi-set. Note that any N-MPDS can be easy transformed into an equivalent one of that form.

5.1 Serialized Executions

Consider an execution $\rho = (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\sigma} (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$ of A starting at the initial configuration (i.e., $(\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) = (\boldsymbol{c}_0, \boldsymbol{m}_0, \boldsymbol{l}_0)$). Let σ_i be the projection of σ on the set δ_i, that is, it is the sequence of transitions executed by thread i along the execution ρ. Then, σ_i can be decomposed uniquely into a sequence of the form: $\tau_1\alpha_2\tau_2 \dots \alpha_j\tau_j\beta$ where, $\tau_1, \dots, \tau_j$ are the transitions in σ_i that remove a task from the multi-set (i.e., $\lambda(\tau_m)$, with $m \in [1..j]$, is of the form $i?a$). Observe that at the configurations of ρ where the transitions τ_m, are executed, the stack of thread i is empty and furthermore the thread i does not hold any locks.

We also decompose β uniquely as $\beta_1\tau'_1\beta_2\tau'_2 \dots \beta_k\tau'_k\gamma$, where τ'_m, $1 \leq m \leq k \in \delta_i$ are the transitions in β which acquire a lock that is not subsequently released. That is, τ'_m is a lock transition on some lock l_m and it is the last transition involving lock l_m in β. We observe that in the configurations of ρ where the transitions τ'_m, $1 \leq m \leq k$ are executed, the set of locks held by thread i must be exactly the set $\{l_r \mid r < m\}$. (Clearly these locks are held, by the definition of τ'_r's. No other lock is held as it would violate the nested locking assumption.)

Thus $\sigma_i = (\tau_1\alpha_1)(\tau_2\alpha_2) \dots (\tau_j\beta_1)(\tau'_1\beta_2) \dots (\tau'_k\gamma)$. We refer to this as the phase decomposition of σ w.r.t. thread i. We further refer to $(\tau_1\alpha_1), (\tau_2\alpha_2), \dots (\tau_{j-1}\alpha_{j-1})$ as *task* phases, $(\tau_j\beta_1)$ as the *boundary* phase and $(\tau'_1\beta_2), (\tau'_2\beta_3) \dots (\tau'_k\gamma)$ as *lock-holding* phases of thread i. Note that there may be no lock-holding phases or even task phases for some threads. A phase of σ is a phase of some thread in σ (and similarly with task, boundary and lock phases).

Given a phase γ in σ we define its index in σ to be the position of the first transition of γ in the sequence σ and write $\mathbf{i}(\gamma)$ to denote this number. Clearly $\mathbf{i}$ defines a linear order on the phases of γ and this allows us to list these phases as $\gamma_1, \gamma_2, \dots, \gamma_m$ where $\mathbf{i}(\gamma_i) < \mathbf{i}(\gamma_j)$ whenever $i < j$. We call this the occurrence ordering of phases. The following Lemma establishes the serialization result we desire.

▶ **Lemma 4.** *Let $\rho = (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\sigma}_A (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$ be a run from the initial configuration. Let $\gamma_1, \gamma_2, \dots \gamma_m$ be the listing of the all phases of σ in the occurrence order. Then, there is a run $\rho' = (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\gamma_1\gamma_2\dots\gamma_m}_A (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$. That is, we can execute the phases of σ atomically (without interleaving) and in their occurrence order.*

5.2 From N-MPDS to 1-MPDS

We are now ready to use the serialization lemma to show that reachability for N-MPDS can be reduced to reachability in an 1-MPDS (or simply MPDS). We will outline the main ideas behind our construction, refining them as we go along, before proceeding to the details.

The multi-set pushdown system that we construct has to maintain the local state of each of the threads. Similarly, it also needs information on the set of locks held by each thread. However notice that by assumption, all valid executions of the system only admits well-nested locks. Hence it is enough to store the first taken lock (that will be released) to know if there are pending locks or if no locks are taken. Thus, each state of the MPDS is of the form $(\boldsymbol{q}, \boldsymbol{l}, z)$, where $\boldsymbol{q}(i)$ denotes the state of thread i and $\boldsymbol{l}(i)$ stores the first lock that was taken (and will be released) by the thread i or is empty if none is taken. The component z holds additional information, such as the identity of the thread being executed (more details will be provided as we go along). We observe that while the N-MPDS has a multi-set per thread, we are only allowed to use a single multi-set in the constructed MPDS. We resolve this by the indexation of each task in the multi-set of the MPDS with the id of the thread it belongs to (i.e., the multi-set alphabet of the MPDS is set to be $\Sigma \times \{1, 2, \dots, N\} \times Y$ where a value

(a, i, y) stands for a value a in the multi-set belonging to thread i, with some additional information y that will be explained later).

A configuration of the MPDS is of the form $(((\boldsymbol{q}, \boldsymbol{l}, z), \alpha), \boldsymbol{M})$ and as explained above it encodes the local states and lock state of a configuration of the N-MPDS. Further, the value $\sum_{y \in Y} \boldsymbol{M}(a, i, y)$ represents $\boldsymbol{m}(i)(a)$, the number of tasks of type a in the multi-set of thread i. Thus, the only unrepresented part of the configuration of the N-MPDS is its collection of stacks, one per thread. Since we have only one stack in the MPDS, we have to manage the simulation of this collection of stacks using this single stack. We shall return to this soon.

Omitting, for the moment, the effect of transitions on the stack (and unexplained parts z and y), the definition of the transitions is easy to see: Simulating a transition of thread i that takes the lock l, involves storing it in the state if it is the very first lock taken by the process (i.e. change the local state component). Unlocking of that transition is handled similarly. Observe that there is no need to keep track of all the taken locks per thread due to the serialization (since the set of available locks at the beginning and end of each phase is already known). To simulate a move posting task a on to thread j, the MPDS posts the value (a, j, y) to its multi-set. Scheduling a task a on thread j corresponds to removing a message of the form (a, j, y) from the multi-set and so on.

We now turn our attention to dealing with the multiple stacks of the N-MPDS. As an immediate consequence of our serialization Lemma, it suffices to simulate the executions of the N-MPDS along executions where each phase is executed atomically. Observe that task phases of any thread begins and ends with the empty stack (in that thread) and empty set of locks. Thus, our MPDS can use its single stack to simulate any sequence of task phases (one after the other). However, the boundary phase does not necessarily end with an empty stack and the lock-holding phases need not begin or end with an empty stack. Consider a sequence of phases of the form $\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6$ where β_1 is a boundary phase of thread i, β_3 and β_6 are lock-holding phases of thread i, β_2 is the boundary phase of task j, β_5 is a lock-holding phase of thread j and β_4 is a phase (of any type) of thread k. The contents of the stack of the thread i must be preserved from the completion of β_1 to the beginning of β_3 and then on to the beginning of β_6, while that of thread j is to be preserved from the completion β_2 to the beginning of β_5. We also need to execute the phase β_4 in between. There is no direct way to achieve this using a single stack. However, we can exploit the fact that there are only a bounded number of boundary and lock-holding phases (actually their number is bounded by the number of threads and the number of locks in the system respectively) to show that we may execute them out of order in a consistent manner using a single stack.

We illustrate the issues involved using the example from the previous paragraph. Suppose β_3, β_5 and β_6 take the locks l_3, l_5 and l_6 respectively and these are never released subsequently. In order to avoid storing the stack contents at the end of a phase (which we cannot), we would like to execute all the lock-holding phases of a task atomically. While it may seem tempting to run β_4 (say if it is a task phase) first, this may not be possible as this task may be created during the execution of β_1 or β_2 or β_3. This kind of causality does not pose a problem as it is handled by the multi-sets. However, we cannot postpone β_4 to completion ahead of the other phases, as β_4 may require the use of lock l_3 which is no longer available after the execution of β_3.

The key idea, is to divide the entire global execution into segments. An initial segment where only task phases are executed (where all locks are available for any phase that is executed) followed by the segment that involves a boundary phase together with task phases, the segment that involves another boundary phase or from the first lock-taking phase to the second (where exactly one lock is unavailable for any phase) together with task phases, the

segment that corresponds to a boundary phase or from the second locking taking phase to the third (where exactly two locks are unavailable for any phase) together with task phases and so on. The number of segments into which any execution breaks up is bounded by the number of locks.

Our MPDS guesses a priori a sequence $w \in ((\mathcal{L} \cup \{\emptyset\}) \times [1..N])^*$ called the guiding sequence (which represents the partition of the global execution into segments). A tuple of the form $(\emptyset, i)$ in the guiding sequence indicates the position of the boundary segment of the thread i. Similarly a tuple of the form (l, i) indicates the positions of the lock-taking segment where the thread i acquires the lock l and never releases it. We call a guiding sequence *valid-guiding-sequence* if all locks occur at-most once in the sequence and if for each thread, the boundary segment precedes the lock-taking segment. Formally, we call a sequence $w = (l_1, i_1) \dots (l_k, i_k) \in ((\mathcal{L} \cup \{\emptyset\}) \times [1..N])^*$ a valid guiding sequence if it has the following properties: 1) For all $i \in [1..N]$, we have $w \downarrow_{(\mathcal{L} \cup \{\emptyset\}) \times \{i\}} \in (\emptyset, i).(\mathcal{L} \times \{i\})^*$, 2) Let $w \downarrow_{\mathcal{L} \times [1..N]} = (s_1, j_1) \cdots (s_m, j_m)$. We have $s_u \neq s_v$ for all $u \neq v$. (i.e. Locks occur only once)

We are only interested in valid guiding sequences and hence, here onwards we will refer to them simply as guiding sequences. We start by guessing a valid sequence and then construct its corresponding MPDS. (Observe that the number of valid sequences is at most exponential in the size of the N-MPDS). As part of the component z of that MPDS, we store the segment number (i.e., the position into the guiding sequence). Initially, the sequence number is initialized to 0 indicating that no part of the guiding sequence is processed. The MPDS begins the simulation with a sequence of task phases that constitute segment 0. When chooses to initiate segment 1. If segment 1 is of the form $(\emptyset, i_1)$ then it picks the thread i_1, executes its boundary phase followed by the execution of all the subsequent phases of thread i_1, each of which must necessarily initiate a segment.

Suppose these segments belonging to the thread i_1 are $1 < j_1 < j_2 \dots < j_m$. Let the entry in the guiding sequence at position j_r be (l_{j_r}, i_1). Then, the MPDS verifies that, while executing the phase of thread i_1 initiating segments j_r, $1 \leq r \leq m$, that the lock l_{j_r} is taken by the first transition and never released, no lock from $\{l | \exists j < j_r, p \in [1..N] : w[j] = (l, p)\}$ is taken during that phase and that any other lock taken is released within the phase. Thus, we execute not just the boundary phase of i_1 but all the subsequent phases of this thread as well. Further, while doing this we ensure that the phases executed out of turn use only the appropriate set of locks available to them if they were executed in the right order. We also ensure that this thread is never scheduled again and this can be taken care of by looking at the current segment number stored in the state. We do not schedule a thread $p \in [1..N]$ if the current segment in the state is j and $w[k] = (\emptyset, p)$ for some $k < j$. We also ensure that the desired final state is reached during such an execution. After this simulation of thread i_1 we increase the segment number to 2 and proceed (if it does not belong to the thread i_1). We do a similar contiguous execution, of the boundary phase and all the subsequent phases of a thread, every time we decide to switch segments and encounter a segment of the form $(\emptyset, i)$. We skip any segment that has been already processed. Such executions ensure the correct usage of the locks using the lock-thread pair sequence. However, we have lost the causal relationship between task creation and execution. For instance, while completing the full execution of i_1 we may create tasks during the execution of segment j_4, but then we can schedule such a task in an earlier segment (with a incorrect lock environment to make it worse). But this problem is solved as follows: we tag the tasks inserted into the multi-set not only with the thread identity but also the segment in which the task is to be scheduled (explaining the third component y in the alphabet of the multi-set alphabet). This target segment of each task is chosen non-deterministically, with a number greater than or equal to

that of the posting transition, tagged with this value and then inserted into the multi-set. Then, a task is picked up from the multi-set only if its segment number tag matches the current segment number. This ensures that tasks are scheduled after their creation (and executed with the right set of locks).

Thus overall the execution of the MPDS may be summarized as the following:

The initialization part of our MPDS initializes the multi-set to hold the initial multi-set symbol for each process. The segment value in the initial state is set to 0. At the beginning of each step the stack is empty. The MPDS can increment the stored segment number in non-deterministic manner. We will refer to the current segment number stored in the state by j. At the beginning of each simulation of a segment (except when $j = 0$), the MPDS removed a task of the form (a, i, j) and executed it as a boundary phase while making sure that the j-th entry in the guiding sequence is of the form $(\emptyset, i)$. Suppose that the sequence of segments corresponding to thread j in the guiding sequence is $(\emptyset, i).(l_1, i) \dots (l_m, i)$, then the boundary phase is first simulated. Subsequently the first locking phase is executed. Then, the MPDS continues the process till the simulation of the last locking phase while making sure that the execution reaches the desired final state of the thread i. On completion, the stack is emptied and we are now allowed to simulate task phases of the from (a, k, j). In this case the task (a, k, j) is executed till the stack is empty again.

During the simulation of a boundary/task phase, the state is tagged with the information on whether the set of locks temporary taken (i.e., will be released) is empty or not. Initially, the state is marked with $\emptyset$ to indicate that there are no locks temporarily held. As soon as a lock is taken (and will be released), the identity of this lock is recorded in the state in place of the empty set $\emptyset$. If the state is already tagged with a lock, subsequent held locks are not stored. When the lock l is released from a state which is tagged with a lock l, the value is reset to $\emptyset$ indicating no more locks are temporarily held at that point.

Observe, that we construct one such automaton per guiding sequence. We run them one after the other to solve the reachability problem for the N-MPDS.

This construction is exponentially large even for a given guiding sequence. We now refine this construction further, making it polynomial — the key idea is to transfer large sections of the control states to the multi-set. This will lead to an automaton whose state space and multi-set alphabet are both polynomial in the size of the original system.

The exponential blow in the state space arises due to the product of the local state spaces and locks that are maintained in the state. During start of a task or a boundary phase, the set of locks held can easily be determined from the guiding sequence. Hence we only need record for the currently active thread, whether the locks held are empty or the identity of the first lock taken for that thread. The key step to eliminate blow up due to product of local state space is to expand the multi-set alphabet to include $\bigcup_{1 \le i \le N} Q_i \times \{i\}$. That is, we keep the current state of each thread, other than the currently executing thread, in the multi-set (transferring the state to the multi-set while switching threads).

With this change we obtain a polynomial sized MPDS, that verifies reachability via runs obeying the given guiding sequence. This results in a Expspace decision procedure per guiding sequence. Since the number of guiding sequences is only exponential, we can run through all candidate sequences and obtain a Expspace procedure overall.

6 Stateless Well-Nested N-MPDS under the Task Locking Policy

Typically asynchronous concurrent software has threads that wait for a task in a while loop. On receipt of a task, they execute it and go back to a waiting state. Motivated by this, we propose a model in which each thread can remove tasks from its multiset only in a particular

state. Formally, let $A = (\Sigma, \mathcal{P}, \mathcal{L})$ be an N-MPDS (as defined in Section 3) and $\boldsymbol{s}$ be a state function that assigns to each index $i \in [1..N]$ a state from Q_i (i.e. $\boldsymbol{s}(i) = q_i$ where q_i is a state of the thread i). We say that A is $\boldsymbol{s}$-stateless if $\delta_i \cap \bigcup_{a\in\Sigma}(Q_i \setminus \{\boldsymbol{s}[i]\} \times \Gamma_i^* \times \{i?a\} \times \Gamma_i^* \times Q_i) = \emptyset$. We say that an N-MPDS A is *stateless* if there is a state function $\boldsymbol{s}$ such that A is $\boldsymbol{s}$-stateless.

In this section, we show that the reachability problem for stateless well-nested N-MPDS under the task locking policy is NP-COMPLETE.

▶ **Theorem 5.** *The reachability problem for stateless well-nested MPDSs under the task-locking policy is* NP-COMPLETE.

The lower bound is proved by reduction from the satisfiability problem of a given 3-SAT formula. In the following, we sketch the proof that establishes the upper-bound. To that aim, let us fix a well-nested N-MPDS $A = (\Sigma, \mathcal{P}, \mathcal{L})$ (as defined in Section 3) under the task-locking policy. Let us assume that the N-MPDS is $\boldsymbol{s}$-stateless for some state function $\boldsymbol{s}$. Further, we assume w.l.o.g. that the only enabled move from the initial state of any thread is a transition that removes a task from the multi-set. In the following, we will show that there is an NP algorithm to solve the reachability problem for A. Towards this, we will show that for any execution of the 1-MPDS, there is a shorter execution that involves only a polynomial number of tasks and reaches the same stack and lock configurations.

▶ **Lemma 6.** *Let $\rho = (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\sigma}_A (\boldsymbol{c'}, \boldsymbol{m'}, \boldsymbol{l'})$ be an execution from the initial configuration. Then, there is another execution $\rho' = (\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{l}) \xrightarrow{\sigma'} (\boldsymbol{c'}, \boldsymbol{m''}, \boldsymbol{l'})$ such that σ' can be decomposed into phases as $\sigma' = \gamma_1 \cdots \gamma_m$ where $m = \mathcal{O}(N \times ((N + |\mathcal{L}|) \times |\Sigma|) + N + |\mathcal{L}|)$.*

We now define K-bounded reachability for an MPDS. Given an MPDS $A = (\Sigma, (Q, \Gamma, \mathcal{O}, \delta, s_1, \alpha_1))$ and a state $q \in Q$, the K-bounded reachability asks if there is an execution $(s_1, w_1, \boldsymbol{M}_1) \xrightarrow{\sigma}_A (q, w, \boldsymbol{M})$, where $(s_1, w_1, \boldsymbol{M}_1)$ is the initial configuration, for some w and $\boldsymbol{M}$ such that $|\sigma \downarrow_{\{1!1(a), 1?a | a\in\Sigma\}}| \leq K$ (i.e., the total number of created and removed tasks is bounded by K). From lemma 6 and the construction in previous section, we get:

▶ **Corollary 7.** *The reachability problem for stateless well-nested N-MPDSs under the task locking policy can be polynomially reduced to the K-bounded reachability for MPDSs, where K is polynomial in the size of the given N-MPDS.*

Proof. The proof of the above corollary uses the same reduction as the one used in Section 5.2. In that construction, the number of operations that remove tasks from the multi-sets of the N-MPDS is the same as the number of operations that remove tasks in the constructed MPDS. By Lemma 6, this number of removed tasks is bounded by a constant K' which is polynomial in the size of the N-MPDS. Furthermore, for any transition (say $(p, \alpha, 1!1(a), \beta, p')$) of the constructed MPDS that creates a task we add to the MPDS another *copy* of the transition that omits the creation of the task (namely a transition of the form $(p, \alpha, \epsilon, \beta, p')$). Since the total number of removed tasks bounds the total number of the tasks that need to be created we get our bound K which can be set to $2K'$. ◀

We will now prove that the K-bounded reachability for MPDSs is in NP. This automatically gives us an NP algorithm for the reachability problem.

▶ **Lemma 8.** *Given an MPDS A and a state q, there is an non-deterministic polynomial time algorithm that decides whether q is K-bounded reachable in A.*

Proof. The NP algorithm starts by guessing the sequence of multi-set operations that create or remove tasks. Observe that the size of such a sequence is bounded by K. Then, the algorithm checks if the guessed sequence is *valid* (i.e., for every prefix of the guessed sequence,

the number of created tasks of a type a is larger that the number of removed tasks of type a). This can be easily checked in polynomial time. Now, if the guessed sequence is valid, the algorithm checks if this sequence can lead to an execution of the MPDS that reaches the state q. This is done by seeing the MPDS $A = (\Sigma, (Q, \Gamma, \mathcal{O}, \delta, s_1, \alpha_1))$ as a pushdown automaton $P = (Q, \Gamma, \mathcal{O}, \delta, s_1, \alpha_1)$ over the alphabet $\{1!1(a), 1?a \mid a \in \Sigma\}$. Finally, checking whether the guessed sequence can lead to an execution of the MPDS that reaches q can be reduced to checking if the guessed sequence can be accepted by the pushdown automaton. ◀

References

1. Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Logical Methods in Computer Science*, 7(4), 2011. doi:10.2168/LMCS-7(4:4)2011.
2. Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. Analyzing asynchronous programs with preemption. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India*, volume 2 of *LIPIcs*, pages 37–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1739.
3. Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2005. doi:10.1007/11539452_36.
4. Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 411–422. ACM, 2011. doi:10.1145/1926385.1926432.
5. Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.
6. Thomas Martin Gawlitza, Peter Lammich, Markus Müller-Olm, Helmut Seidl, and Alexander Wenner. Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2011. doi:10.1007/978-3-642-18275-4_15.
7. Ranjit Jhala and Rupak Majumdar. Interprocedural analysis of asynchronous programs. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 339–350. ACM, 2007. doi:10.1145/1190216.1190266.
8. Vineet Kahlon, Franjo Ivancic, and Aarti Gupta. Reasoning about threads communicating via locks. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 505–518. Springer, 2005. doi:10.1007/11513988_49.
9. Peter Lammich and Markus Müller-Olm. Conflict analysis of programs with procedures, dynamic thread creation, and monitors. In María Alpuente and Germán Vidal, editors, *Static Analysis, 15th International Symposium, SAS 2008, Valencia, Spain, July 16-18, 2008. Proceedings*, volume 5079 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2008. doi:10.1007/978-3-540-69166-2_14.

10 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005. doi:10.1007/978-3-540-31980-1_7.

11 G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000. doi:10.1145/349214.349241.

12 Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2006. doi:10.1007/11817963_29.

Modulo Counting on Words and Trees*†

Bartosz Bednarczyk[1] and Witold Charatonik[2]

1 **Computer Science Department, ENS Paris-Saclay, Cachan, France; and Institute of Computer Science, University of Wrocław, Wrocław, Poland**
bartosz.bednarczyk@ens-paris-saclay.fr

2 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
wch@cs.uni.wroc.pl

Abstract

We consider the satisfiability problem for the two-variable fragment of the first-order logic extended with modulo counting quantifiers and interpreted over finite words or trees. We prove a small-model property of this logic, which gives a technique for deciding the satisfiability problem. In the case of words this gives a new proof of ExpSpace upper bound, and in the case of trees it gives a 2-ExpTime algorithm. This algorithm is optimal: we prove a matching lower bound by a generic reduction from alternating Turing machines working in exponential space; the reduction involves a development of a new version of tiling games.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases satisfiability, trees, words, two-variable logic, modulo quantifiers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.12

1 Introduction

Two-variable logics. Two-variable logic, FO^2, is one of the most prominent decidable fragments of first-order logic. It is important in computer science because of its decidability and connections with other formalisms like modal, temporal and description logics or query languages. The satisfiability problem for FO^2 is NExpTime-complete and satisfiable formulas have models of exponential size. In this paper we are interested in extensions of FO^2 interpreted over finite words and trees. It is known that FO^2 over words can express the same properties as unary temporal logic [9] and FO^2 over trees is precisely as expressive as the navigational core of XPath, a query language for XML documents [16]. The satisfiability problem for FO^2 over words is shown to be NExpTime-complete in [9], and over trees – ExpSpace-complete in [4]. Recently it was shown that these complexities do not change[7, 3] if counting quantifiers of the form $\exists^{\leq k}, \exists^{\geq k}$ are added.

Modulo counting quantifiers. First-order logic has a limited expressive power. For example, it cannot express even such simple quantitative properties as parity. To overcome this problem, Wolfgang Thomas with his coauthors introduced in the 80s of the last century *modulo counting quantifiers* of the form "there exists a mod b elements x such that ...". A survey of results in first order logic extended with modulo counting quantifiers can be found in [22]. Recent research in this area involves a study of definability of regular languages on words and its connections to algebra [22], [23]; definable languages of $(\mathbb{N}, +)$ [19], [14]; equivalences

* Supported by the Polish National Science Centre grant No. 2016/21/B/ST6/01444.
† A full version of the paper is available at http://arxiv.org/abs/1710.05582.

of finite structures [17]; definable tree languages [18], [5]; locality [11], [12]; extensions of Linear Temporal Logic [1], [21], complexity of the model-checking problem [12], [6]. Not much is known about the complexity of the satisfiability problem for this logic. The only work that we are aware of is [15], which proves ExpSpace-completeness of the satisfiability problem for FO^2 with modulo counting quantifiers over finite words. On the other hand, a simple adaptation of automata techniques for deciding WS1S or WS2S gives a non-elementary decision procedure for the satisfiability problem of full first-order logic with modulo quantifiers over words or ranked trees.

Our contribution. We consider the satisfiability problem for FO^2 with modulo counting quantifiers interpreted over finite words and trees. We provide an alternative to [15] proof of ExpSpace upper bound for the case of words. This proof is based on a small-model property of this logic and can be extended to the case of ordered and unranked trees. By ordered, we mean that the list of children of any node is ordered by the sibling relation, and by unranked, that there is no limit on the number of children. We prove that the case of such trees is 2-ExpTime-complete. For the upper bound, we again prove a small-model theorem and then give an alternating algorithm that decides the problem in exponential space. Since AExpSpace = 2-ExpTime, this gives the desired upper bound. With some obvious modifications this algorithm can be also applied to unordered or ranked trees. For the lower bound, we develop a new version of tiling games that can encode computations of alternating Turing machines working in exponential space, and can be encoded in the logic. In our encoding we do not use ordering of children, and the number of children of any node is bounded by 2, which shows that already the case of unordered and ranked trees is 2-ExpTime-hard.

Due to space limits some proofs and encodings are omitted. They can be found in the full version of the paper [2].

2 Preliminaries

Tuples and modulo remainders. By a k-tuple of numbers we mean an element of the set $\mathbb{N}^k$. By $\vec{0}_k$ and $\vec{1}_k$ we will denote k-tuples consisting of, respectively, only zeros and only ones. We denote the i-th element of a k-tuple t by $\pi_i(t)$. We will often drop the subscript k if the dimension k is clear from the context.

Consider a finite set X, a number $k \in \mathbb{N}$ and a mapping f from X to $\mathbb{N}^k$. We say that f is *zero*, if for all $x \in X$ we have $f(x) = \vec{0}_k$. We say that f is a *singleton*, if there exists a unique argument $x \in X$ such that $f(x) = \vec{1}_k$ and for all other arguments the function f is zero.

Let n be a positive integer. By $\mathbb{Z}_n$ we denote the set of all remainders modulo n, that is the set $\{0, 1, \ldots, n-1\}$. We denote by r_n the remainder function modulo n, that is $r_n : \mathbb{N} \to \mathbb{Z}_n$, $r_n(x) = x \bmod n$.

Syntax and semantics. We refer to structures with fraktur letters, and to their universes with the corresponding Roman letters. We always assume that structures have non-empty universes. We work with signatures of the form $\tau = \tau_0 \cup \tau_{nav}$, where τ_0 is a set of unary relational symbols, and $\tau_{nav} \subseteq \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+, \leq, succ\}$ is the set of *navigational* binary symbols, which will be interpreted in a special way, depending on which kind of structures, words or trees, are considered as models.

We define the two-variable fragment of first-order logic with modulo counting quantifiers $\mathrm{FO}^2_{\mathrm{MOD}}$ as the set of all first-order formulas (over the signature τ), featuring only the variables x and y, extended with modulo counting quantifiers of the form $\exists^{\leq k,l}, \exists^{=k,l}$ and $\exists^{\geq k,l}$, where $l \in \mathbb{Z}_+$ is a positive integer and $k \in \mathbb{Z}_l$. The formal semantics of such quantifiers is as follows: a formula $\exists^{\bowtie k,l}\, y\, \varphi(x,y)$, where $\bowtie \in \{\leq, =, \geq\}$, is true at element a of a structure $\mathfrak{M}$, in symbols $\mathfrak{M}, a \models \left(\exists^{\bowtie k,l}\, y\, \varphi(x,y)\right)$ if and only if $r_l\,(\,|\{b \in M : \varphi[a,b]\}|\,) \bowtie k$. When measuring the length of formulas we will assume binary encodings of numbers k and l in superscripts of modulo quantifiers.

In [15] the authors use a different notation for modulo counting quantifiers in their logic $\mathrm{FO}\textsc{mod}^2[<]$. It seems that the main difference is that they require equality in place of our $\bowtie$ comparison. The use of $\leq$ and $\geq$ operators makes the logic exponentially more succinct. For example the formula $\exists^{\geq 2^n, 2\cdot 2^n} x\; p(x)$ has length $\mathcal{O}(n)$, while an equivalent formula in $\mathrm{FO}\textsc{mod}^2[<]$ has the form $\bigvee_{i=2^n}^{2\cdot 2^n} \exists^{=i, 2\cdot 2^n} x\; p(x)$ and its length is $\Omega(2^n \cdot n)$.

We write $\mathrm{FO}^2_{\mathrm{MOD}}[\tau_{nav}]$, to denote that the only binary symbols that are allowed in signatures are from τ_{nav}. We will work with two logics: $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ and $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$. The former is interpreted over finite words, where $\leq$ denotes the linear order over word positions and *succ* is the successor relation. The latter logic is interpreted over finite, unranked and ordered trees. The interpretation of the symbols from τ_{nav} is the following: $\downarrow$ is interpreted as the child relation, $\rightarrow$ as the right sibling relation, and $\downarrow^+$ and $\rightarrow^+$ as their respective transitive closures. We read $u \downarrow w$ as "w is a *child* of u" and $u \rightarrow w$ as "w is the *right sibling* of u". We will also use other standard terminology like *ancestor*, *descendant*, *preceding-sibling*, *following-sibling*, etc. In both cases of words and trees all elements of the universe can be labeled by an arbitrary number of unary predicates from τ_0.

Normal form. As usual when working with two-variable logic, it is convenient to use so-called Scott normal form [20]. The main feature of such form is that nesting of quantifiers is restricted to depth two. Below we adapt this notion to the setting of $\mathrm{FO}^2_{\mathrm{MOD}}$.

▶ **Definition 1.** We say that a formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}$ is in *normal form*, if

$$\varphi = \forall x \forall y\; \chi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x \exists y\; \chi_i(x,y) \wedge \bigwedge_{j=1}^{m} \forall x \exists^{\bowtie_j k_j, l_j} y\; \psi_j(x,y)$$

where $\bowtie_i \in \{\leq, \geq\}$, each l_j is a positive integer and each $k_j \in \mathbb{Z}_{l_j}$ and all χ, χ_i and ψ_j formulas are quantifier-free. We require both n and m parameters to be non-zero.

The following lemma is proved by a routine adaptation of a normal form proof from [10].

▶ **Lemma 2.** *Let φ be a formula from $\mathrm{FO}^2_{\mathrm{MOD}}$ over a signature τ. There exists a polynomially computable $\mathrm{FO}^2_{\mathrm{MOD}}$ normal form formula φ' over signature τ', consisting of τ and polynomially many additional unary symbols, such that φ and φ' are equisatisfiable.*

In the next sections, when a normal form formula φ is considered we always assume that it is as in Definition 1. In particular we allow ourselves, without explicitly recalling the shape of φ, to refer to its parameters n, m, components χ, χ_i, ψ_j and numbers k_j, l_j given in modulo counting quantifiers.

Consider a conjunct $\forall x \exists y\; \chi_i(x,y)$ from an $\mathrm{FO}^2_{\mathrm{MOD}}$ normal form formula φ. Let $\mathfrak{M}$ be its finite model and let $v \in M$ be an arbitrary element from the structure. Then an element $w \in M$ such that $\mathfrak{M} \models \chi_i[v,w]$ is called a *χ_i-witness* for v. Analogously, we will speak about ψ_j-witnesses for modulo conjuncts or simply witnesses if a formula is clear from the context.

Order formulas. Let us call *order formulas* the formulas specifying relative positions of pairs of elements in a structure with respect to the binary navigational predicates.

In the case of words, when $\tau_{nav} = \{\leq, succ\}$, there are five possible order formulas: $x{=}y$, $succ(x,y)$, $succ(y,x)$, $x{\neq}y \wedge \neg succ(x,y) \wedge x{\leq}y$ and $x{\neq}y \wedge \neg succ(y,x) \wedge y{\leq}x$. We denote them respectively as $\theta_{=}, \theta_{+1}, \theta_{-1}, \theta_{\ll}, \theta_{\gg}$. Let Θ_{words} be the set of these five formulas.

In the case of trees, when $\tau_{nav} = \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+\}$, we use $x{\not\sim}y$ to abbreviate the formula stating that x and y are in *free position*, i.e., that they are related by none of the navigational binary predicates available in the signature. There are ten possible order formulas: $x{\downarrow}y$, $y{\downarrow}x$, $x{\downarrow^+}y \wedge \neg(x{\downarrow}y)$, $y{\downarrow^+}x \wedge \neg(y{\downarrow}x)$, $x{\rightarrow}y$, $y{\rightarrow}x$, $x{\rightarrow^+}y \wedge \neg(x{\rightarrow}y)$, $y{\rightarrow^+}x \wedge \neg(y{\rightarrow}x)$, $x{\not\sim}y$, $x{=}y$. They are denoted, respectively, as: $\theta_{\downarrow}$, $\theta_{\uparrow}$, $\theta_{\downarrow\downarrow^+}$, $\theta_{\uparrow\uparrow^+}$, $\theta_{\rightarrow}$, $\theta_{\leftarrow}$, $\theta_{\rightrightarrows^+}$, $\theta_{\leftleftarrows^+}$, $\theta_{\not\sim}$, $\theta_{=}$. Let Θ_{trees} be the set of these ten formulas. We will use symbol Θ to denote either Θ_{words} or Θ_{trees} if the type of structure is not important or clear from context.

Types and local configurations. Following a standard terminology, an *atomic* 1*-type* over a signature τ is a maximal satisfiable set of atoms or negated atoms involving only the variable x. Usually we identify a 1-type with the conjunction of all its elements. We will denote by $\boldsymbol{\alpha}$ the set of all 1-types over τ. Note that the size of $\boldsymbol{\alpha}$ is exponential in the size of τ. For a given τ-structure $\mathfrak{M}$, and an element $v \in M$ we say that v *realizes* a 1-type α, if α is the unique 1-type such that $\mathfrak{M} \models \alpha[v]$. We denote by $\mathrm{tp}^{\mathfrak{M}}(v)$ the 1-type realized by v.

A 1-type provides us only information about a single element of a model. Below we generalize this notion such that for each element of a structure it provides us also some information about surrounding elements and their 1-types.

▶ **Definition 3.** An $(l_1, l_2, \ldots, l_m)$-full type $\overline{\alpha}$ over a signature τ is a function of the type $\overline{\alpha} : \Theta \to \boldsymbol{\alpha} \to \{0,1\} \times \mathbb{Z}_{l_1} \times \mathbb{Z}_{l_2} \times \ldots \times \mathbb{Z}_{l_m}$ (a function that takes an order formula from Θ and returns a function that takes a 1-type from $\boldsymbol{\alpha}$ and returns a tuple of integers). Additionally, we require that all $(l_1, l_2, \ldots, l_m)$-full types satisfy the following conditions:

- $\overline{\alpha}(\theta_{=})$ is a singleton,
- $\overline{\alpha}(\theta)$ is either zero or singleton for $\theta \in \{\theta_{+1}, \theta_{-1}, \theta_{\rightarrow}, \theta_{\uparrow}, \theta_{\leftarrow}\}$ and
- if $\overline{\alpha}(\theta_{+1})$ (respectively $\overline{\alpha}(\theta_{-1}), \overline{\alpha}(\theta_{\rightarrow}), \overline{\alpha}(\theta_{\downarrow}), \overline{\alpha}(\theta_{\leftarrow}), \overline{\alpha}(\theta_{\uparrow})$) is zero then also the function $\overline{\alpha}(\theta_{\gg})$ (respectively $\overline{\alpha}(\theta_{\ll}), \overline{\alpha}(\theta_{\rightrightarrows^+}), \overline{\alpha}(\theta_{\downarrow\downarrow^+}), \overline{\alpha}(\theta_{\leftleftarrows^+}), \overline{\alpha}(\theta_{\uparrow\uparrow^+})$) is zero.

In the following, if the numbers $l_1, l_2, \ldots, l_m$ are clear from the context or not important, we will be omitting them.

Let us briefly describe the idea behind the notion of full types. Ideally, for a given element v of a structure $\mathfrak{M}$, we would like to know the exact number of occurrences of each 1-type on each relative position. However, to keep the memory usage under control, we store only a summary of this information. For the purpose of $\forall\exists$ conjuncts of a formula in normal form, it is enough to know if a type α occurs at least once in a given relative position. This information is stored in the 0-1 part of a full type. Additionally, for the purpose of $\exists^{\bowtie_j k_j, l_j}$ conjuncts, we store the remainders of the total numbers of occurrences of each 1-type α modulo each of l_j appearing in modulo quantifiers.

▶ **Definition 4.** Let $\mathfrak{M}$ be a τ-structure and v an arbitrary element from M. We will denote by $(l_1, l_2, \ldots, l_m)$–$\mathrm{ftp}^{\mathfrak{M}}(v)$ the unique $(l_1, l_2, \ldots, l_m)$-full type *realized* by v in $\mathfrak{M}$, i.e., the $(l_1, l_2, \ldots, l_m)$-full type $\overline{\alpha}$ such that for all order formulas $\theta \in \Theta$ and for all atomic 1-types α, the value of $\overline{\alpha}(\theta)(\alpha)$ is equal to the tuple $(cut_1(W), r_{l_1}(W), r_{l_2}(W), \ldots, r_{l_m}(W))$, where W is the number of occurrences of elements of type α in the relative position described by θ, namely

$$W = |\{w \in M : \mathfrak{M} \models \theta[v,w] \wedge \mathrm{tp}^{\mathfrak{M}}(w) = \alpha\}|.$$

and $cut_1(W)$ is 0 if W is empty and 1 otherwise.

The following definition and lemma are basic tools used in the proofs of small-model properties $\mathrm{FO}^2_{\mathrm{MOD}}$.

▶ **Definition 5** (φ-consistency). Let φ be a $\mathrm{FO}^2_{\mathrm{MOD}}$ formula in normal form and let α be the unique 1-type satisfying $\overline{\alpha}(\theta_=)(\alpha) = \vec{1}$. We say that a $(l_1, l_2, \ldots, l_m)$-full type $\overline{\alpha}$ is *φ-consistent*, if it satisfies the following conditions:

1. It does not violate the $\forall\forall$ subformula i.e. for all order formulas $\theta \in \Theta$ and for all 1-types β such that $\overline{\alpha}(\theta)(\beta) \neq \vec{0}$, the implication $\alpha(x) \wedge \beta(y) \wedge \theta(x,y) \models \chi(x,y)$ holds.
2. There is a witness for each $\forall\exists$ conjunct of φ. Formally, consider a number $1 \leq i \leq n$ and a subformula $\forall x \exists y\ \chi_i(x,y)$. We require that there exists a 1-type β and an order formula $\theta \in \Theta$ such that $\overline{\alpha}(\theta)(\beta) \neq \vec{0}$ and the logical implication $\alpha(x) \wedge \beta(y) \wedge \theta(x,y) \models \chi_i(x,y)$ holds.
3. Each modulo conjunct has the right number of witnesses. Consider a number $1 \leq j \leq m$ and a subformula $\forall x \exists^{\bowtie_j k_j, l_j} y\ \psi_j(x,y)$. We require that the remainder modulo l_j of the number of witnesses encoded in the full-type satisfies the inequality $\bowtie_j k_j$. Formally,

$$r_{l_j}\left(\sum_{\theta\in\Theta,\beta\in\boldsymbol{\alpha}\,:\,\alpha(x)\wedge\beta(y)\wedge\theta(x,y)\models\psi_j(x,y)} \pi_{j+1}\left(\overline{\alpha}(\theta)(\beta)\right)\right) \bowtie_j k_j.$$

A proof of the following lemma is a straightforward unfolding of Definitions 4 and 5 and the definition of the semantics of $\mathrm{FO}^2_{\mathrm{MOD}}$.

▶ **Lemma 6.** *Assume that a formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}$ in normal form is interpreted over finite words or trees. Then $\mathfrak{M} \models \varphi$ if and only if every $(l_1, l_2, \ldots, l_m)$-full type realized in $\mathfrak{M}$ is φ-consistent.*

Let φ be $\mathrm{FO}^2_{\mathrm{MOD}}$ formula in normal form. For proving the upper bounds in the next sections, we will estimate the size of a model by the following function. We define $\mathfrak{f}(\varphi)$ as the function, which for a given formula returns the total number of $(l_1, l_2, \ldots, l_m)$-full types over the signature of φ. To be precise, $\mathfrak{f}(\varphi)$ is equal to $(2l_1 l_2 \ldots l_m)^{|\Theta||\boldsymbol{\alpha}|}$. Note that $\mathfrak{f}(\varphi)$ is doubly exponential in the size of the formula φ.

3 Finite words

In this section we focus our attention on the case of finite words. We give an alternative proof of ExpSpace upper bound for the satisfiability problem of $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, \mathit{succ}]$. Originally this result was proved in [15] by a reduction to unary temporal logic with modulo counting operators. Here we give a direct algorithm dedicated to $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, \mathit{succ}]$, based on a small model property that we prove. The advantage of the method is that it allows an extension to the case of trees and other extensions (e.g., an incorporation of counting quantifiers of the form $\exists^{\leq k}, \exists^{\geq k}$ is quite obvious).

Small model property. We start by proving that every satisfiable formula in $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, \mathit{succ}]$ has a small model. The proof technique is similar to the pumping lemma known from the theory of finite word automata.

▶ **Lemma 7.** *Every normal form $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, \mathit{succ}]$ formula satisfiable over finite words has a model $\mathfrak{W}$ of size bounded by $\mathfrak{f}(\varphi)$.*

Proof. Consider a satisfiable formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\leq, \mathit{succ}]$ and assume that its model $\mathfrak{W}$ is a word longer than $\mathfrak{f}(\varphi)$. We will show that we can remove some subword from $\mathfrak{W}$ and

Procedure 1: Satisfiability test for $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$

Input: Formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ in normal form.

```
MaxLength := f(φ)                                  // Maximal length of a model from Lemma 7
CurrentPosition := 0
guess a full type ᾱ s.t. ᾱ(θ≪) and ᾱ(θ−1) are zero    // Type of the first position
if not is-φ-consistent(ᾱ) then reject                  // See Definition 5
while CurrentPosition < MaxLength do
    if both ᾱ(θ≫) and ᾱ(θ+1) are zero then accept   // Type of the last position
    guess a full type β̄                                // Type of the successor
    if not is-φ-consistent(β̄) then reject              // See Definition 5
    if not is-valid-successor(β̄, ᾱ) then reject
    ᾱ := β̄
    CurrentPosition := CurrentPosition + 1
reject
```

obtain a shorter model. By repeating this process, we will finally obtain a model of φ with a required size.

By the pigeonhole principle there exist two positions $u, v \in W$ with equal full-types. Let $\mathfrak{W}'$ be a word obtained from $\mathfrak{W}$ by removing all letters from positions between u and v and collapsing u and v into a single position. Observe that since full types of u and v are equal, for all j the remainders modulo l_j of the total number of removed 1-types on positions between u and v are equal to 0. Note also that due to presence of 0-1 part in the definition of a full type, all unique 1-types realized by the structure survive the surgery. Therefore, all full types in $\mathfrak{W}$ remain unchanged. Since all these full types were φ-consistent in $\mathfrak{W}$, they are also φ-consistent in $\mathfrak{W}'$. As a consequence of Lemma 6, the word $\mathfrak{W}'$ is indeed a model of φ, as expected. ◀

Algorithm. Now we are ready to present an ExpSpace algorithm for solving the satisfiability for $\mathrm{FO}^2_{\mathrm{MOD}}[\leq, succ]$ interpreted over finite words. We assume that the input formula φ is in normal form. Since models of φ can have doubly-exponential length, we cannot simply guess a complete intended model. To overcome this difficulty, we guess the model on the fly, letter by letter. For each position of the guessed word we guess its full type and after checking some consistency conditions described in Definition 5, we verify if it can be linked with the previous position. This way we never have to store in memory more than two full types. Since the size of a full type is bounded exponentially in $|\varphi|$, the whole procedure runs in ExpSpace. We accept the input, if the guessing process ends after at most $\mathfrak{f}(\varphi)$ steps.

To avoid presentational clutter in the description of our algorithm we omit the details of two simple tests *is-valid-successor* and *is-φ-consistent*. The former takes two full types, $\overline{\beta}$ and $\overline{\alpha}$, and checks if the position of the type $\overline{\alpha}$ can indeed have a successor of type $\overline{\beta}$. It can be easily done by comparing the total number of 1-types on each relative position stored in both types. The latter test takes one full type $\overline{\alpha}$ and checks if it satisfies the conditions described in Definition 5.

The correctness of the algorithm above is guaranteed by the following lemma.

▶ **Lemma 8.** *Procedure 1 accepts its input φ if and only if φ is satisfiable.*

4 Finite trees

In this section we prove the main result of this paper, which is the following theorem. It is a direct consequence of Theorems 12 and 16 below.

▶ **Theorem 9.** *The satisfiability problem for* $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ *interpreted over finite trees is 2-*ExpTime*-complete.*

4.1 Upper bound

We start by proving the 2-ExpTime upper bound. As in previous section, this is done by showing first a small model property and then an algorithm. The small model property is crucial in the proof of correctness of the algorithm. Actually, having defined the notion of the full type, the technique here is a rather straightforward combination of the technique from previous section and from [4]. The main difficulty here was to come up with the right notion of a full type.

4.1.1 Small model property

We demonstrate the small-model property of the logic $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ by showing that every satisfiable formula φ has a tree model of depth and degree bounded by $\mathfrak{f}(\varphi)$. This is done by first shortening $\downarrow$-paths and then shortening the $\rightarrow$-paths, as in the proof of Lemma 7.

▶ **Theorem 10** (Small model theorem)**.** *Let* φ *be a normal form* $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ *formula. If* φ *is satisfiable then it has a a tree model in which every path has length bounded by* $\mathfrak{f}(\varphi)$ *and every vertex has degree bounded by* $\mathfrak{f}(\varphi)$*.*

Proof. Let $\mathfrak{T}$ be a model of φ. First we show how to shorten $\downarrow$-paths in $\mathfrak{T}$. Assume that there exists a $\downarrow$-path in $\mathfrak{T}$ longer than $\mathfrak{f}(\varphi)$. Thus, by the pigeonhole principle, there are two nodes u and v on this path such that v is a descendant of u and $\mathrm{ftp}^{\mathfrak{T}}_{\varphi}(u) = \mathrm{ftp}^{\mathfrak{T}}_{\varphi}(v)$. Consider the tree $\mathfrak{T}'$, obtained by removing the subtree rooted at u and replacing it by the subtree rooted at v.

Observe that for all j the remainders modulo l_j of the total number of removed vertices is equal to 0. Additionally, all unique full types survive the surgery. Thus, all full types are retained from the original tree, so they are φ-consistent and by Lemma 6 the tree $\mathfrak{T}'$ is a model of φ. By repeating this process we get a tree with all $\downarrow$-paths shorter than $\mathfrak{f}(\varphi)$.

Shortening the $\rightarrow$-paths is done in a similar way. Assume that there exists a node v with branching degree greater than $\mathfrak{f}(\varphi)$. Then there exist two children u, w of v, with equal full types. Let $\mathfrak{T}''$ be a tree obtained by removing all vertices between u and w (excluding u and including w) together with subtrees rooted at them. Again, the remainders modulo l_j of the total number of removed vertices are all equal to 0, and all unique types survive the surgery, so all full types are retained from $\mathfrak{T}$ and thus φ-consistent. Therefore the tree $\mathfrak{T}''$ is a model of φ. We repeat this process until we get a tree with desired branching. ◀

4.1.2 Algorithm

In this section, we design an algorithm to check if a given $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ formula φ interpreted over finite trees is satisfiable. By Lemma 2 we can assume that the input formula φ is given in normal form. Then, by Theorem 10, we can turn our attention to trees with degree and path length bounded by $\mathfrak{f}(\varphi)$. Recall that $\mathfrak{f}(\varphi)$ is doubly-exponential in $|\varphi|$.

Let us describe the core ideas of the algorithm. It works in alternating exponential space. Since AExpSpace = 2-ExpTime, it can be translated to an algorithm working in 2-ExpTime. The algorithm starts by guessing the full type of the root and then it calls a procedure that builds a tree with the given full type of the root.

The procedure called on a vertex v guesses on the fly the children of v, starting with the leftmost child and storing in the memory at most two children at the same time. While guessing a child v', it checks that its type is φ-consistent and that it can be correctly linked to v and to its left sibling (if it exists). Then the procedure is called recursively on v' to build the subtree rooted at v', but the recursive call is done in parallel, exploiting the power of alternation. The process is continued in the same way, until the bottom of the tree is reached or the height of the constructed tree exceeds $\mathfrak{f}(\varphi)$.

There is one more point of the algorithm, namely, we have to make sure that the types of children and their descendants, guessed on the fly, are consistent with the information stored in their parent's full type, in particular with $\theta_{\downarrow}$ and $\theta_{\downarrow\downarrow^+}$ components. To handle the first of them, we can simply compute the union of all $\theta_{=}$ components of children, which can be done during the guessing process. Analogously, to handle the second case, we simply compute the union of all $\theta_{\downarrow}$ and $\theta_{\downarrow\downarrow^+}$ components of all children's full types.

Below we present a pseudocode of the described procedure. Similarly to the case of finite words, we omit the details of obvious methods for checking consistency. For calculating the union of full-types mentioned in the previous paragraph, we employ $\oplus$ operation defined as follows. For given $(l_1, l_2, \ldots, l_m)$-full types $\overline{\alpha}$ and $\overline{\beta}$, the result of $\overline{\alpha} \oplus \overline{\beta}$ is a $(l_1, l_2, \ldots, l_m)$-full type $\overline{\gamma}$ such that for all order formulas $\theta \in \Theta$ and all 1-types $\alpha \in \boldsymbol{\alpha}$, the following condition holds:

$$\overline{\gamma}(\theta)(\alpha) = \left(\max\left(\pi_0(\overline{\alpha}(\theta)(\alpha)), \pi_0(\overline{\beta}(\theta)(\alpha))\right), R_1, R_2, \ldots, R_m\right),$$

where $R_i = r_{l_i}(\pi_i(\overline{\alpha}(\theta)(\alpha)) + \pi_i(\overline{\beta}(\theta)(\alpha)))$.

The following lemma guarantees the correctness of the algorithm and leads directly to the main result of this section.

▶ **Lemma 11.** *Procedure 3 accepts its input formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ if and only if φ is satisfiable over finite trees.*

▶ **Theorem 12.** *The satisfiability problem for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ interpreted over finite trees is in 2-ExpTime.*

4.2 Lower bound

In this section we prove that the satisfiability of $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ is 2-ExpTime-hard. We exploit here the fact that 2-ExpTime =AExpSpace and provide a generic reduction from AExpSpace. This is done in two steps. First we translate computations of alternating Turing machines to winning strategies in (our version of) tiling games. These strategies are then encoded as trees and their existence is translated to the satisfiability problem in $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$.

Alternating Turing machines. An alternating Turing machine is Turing machine whose set of states contains *existential* states and *universal* states. A input word is *accepted* by such a machine if the initial configuration *leads to acceptance*, where leading to acceptance is defined recursively as follows: accepting configurations (i.e., configurations containing

Procedure 2: Building a subtree rooted at given node

Input: Formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ in normal form, full type $\overline{\alpha}$ of a starting node, and current level $\mathrm{Lvl} \in \mathbb{N}$.

if not is-φ-consistent$(\overline{\alpha})$ **then reject** // See Definition 5
if $\mathrm{Lvl} \geq \mathfrak{f}(\varphi)$ **then reject** // Path too long
if $\overline{\alpha}(\theta_\downarrow)$ is zero **then accept** // Last node on the path
Guess the degree $\mathrm{Deg} \in [1, \mathfrak{f}(\varphi)]$ of a node
Guess the full type $\overline{\beta}$ of the leftmost child and check if its a valid leftmost son of $\overline{\alpha}$
$O_{\theta_\downarrow} := \overline{\beta}(\theta_=)$ // Types of children guessed so far
$O_{\theta_{\downarrow\downarrow^+}} := \overline{\beta}(\theta_\downarrow) \oplus \overline{\beta}(\theta_{\downarrow\downarrow^+})$ // Types of descendants guessed so far
while $Deg > 1$ **do**
  Run in parallel Procedure 2 on $(\varphi, \overline{\beta}, \mathrm{Lvl}+1)$ // Alternation here
  Guess a full type $\overline{\gamma}$ of the right brother of $\overline{\beta}$ and check consistency with $\overline{\alpha}$
  $O_{\theta_\downarrow} := O_{\theta_\downarrow} \oplus \overline{\gamma}(\theta_=)$, $O_{\theta_{\downarrow\downarrow^+}} := O_{\theta_{\downarrow\downarrow^+}} \oplus \overline{\gamma}(\theta_\downarrow) \oplus \overline{\gamma}(\theta_{\downarrow\downarrow^+})$ // Updating obligations
  $\overline{\beta} := \overline{\gamma}$, $\mathrm{Deg} := \mathrm{Deg} - 1$
Run in parallel Procedure 2 on $(\varphi, \overline{\beta}, \mathrm{Lvl}+1)$ // Last child
if $\overline{\beta}(\theta_\rightarrow)$ is not zero **then reject** // Not valid last node on $\rightarrow$-path.
if $\overline{\alpha}(\theta_\downarrow) = O_{\theta_\downarrow}$ and $\overline{\alpha}(\theta_{\downarrow\downarrow^+}) = O_{\theta_{\downarrow\downarrow^+}}$ **then accept else reject**

Procedure 3: Satisfiability test for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$

Input: Formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+, \rightarrow, \rightarrow^+]$ in normal form.

guess a full type $\overline{\alpha}$ s.t. $\overline{\alpha}(\theta_\uparrow)$ is zero // Type of the root
Run Procedure 2 on $(\varphi, \overline{\alpha}, 1)$

accepting state) lead to acceptance; an existential configuration leads to acceptance if there exists its successor configuration that leads to acceptance; a universal configuration leads to acceptance if all its successor configurations lead to acceptance. More details can be found in [2].

4.2.1 Tilling Games

Corridor tiling games, aka rectangle tiling games [8] provide a well-known technique for proving lower bounds in space complexity. Here we develop our own version of these games that is able to encode alternating Turing machines from previous section and to be encoded in $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$.

By a tiling game we understand a tuple of the form $\langle C, T_0, T_1, n, \langle t_0, \ldots, t_n\rangle, \square, L\rangle$, where C is a finite set of *colors*; $T_0, T_1 \subseteq C^4$ are two sets of *tiles* (these two sets are not meant to be disjoint); n is a natural number; $\langle t_0, \ldots, t_n\rangle$ is an *initial* sequence of $n+1$ tiles; $\square \in C$ is a special color called *white*; $L \subseteq T_0 \cup T_1$ is a set of tiles allowed in the *last* row.

We think of a tile $\langle a, b, c, d\rangle \in C^4$ as of a square consisting of four smaller squares, colored respectively with colors a, b, c and d (see Figure 1). In the following we will require that adjacent tiles have matching colors, both horizontally and vertically. Formally, we define the horizontal adjacency relation $H = \{\langle\langle a, b, c, d\rangle, \langle b, e, d, f\rangle\rangle \mid a, b, c, d, e, f, \in C\}$ and the vertical adjacency relation $V = \{\langle\langle a, b, c, d\rangle, \langle c, d, e, f\rangle\rangle \mid a, b, c, d, e, f, \in C\}$. We define a *correctly tiled corridor* to be a rectangle of size $k \times 2^n$ for some $k \in \mathbb{N}$ filled with tiles, with all horizontally adjacent tiles in H and all vertically adjacent tiles in V, with first row starting

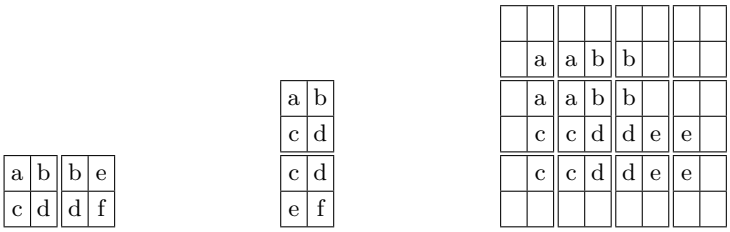


Figure 1 Horizontally adjacent tiles, vertically adjacent tiles, and a correct tiling

with tiles $t_0, \ldots, t_n$ and padded out with white tiles, with last row built only from tiles in L, and with all edges being white. Figure 1 shows an example of correctly tiled corridor for $n = 2$ and initial tiles $\langle \square, \square, \square, a\rangle, \langle \square, \square, a, b\rangle, \langle \square, \square, b, \square\rangle$.

Consider the following game. There are two players, *Prover* (also called the *existential* player) and *Spoiler* (also called the *universal* player). The task of Prover is to construct a correct tiling; the task of Spoiler is to prevent Prover from doing this. At the beginning they are given the initial row $t_0, \ldots, t_n, (\square^4)^{2^n-n-1}$. In each move the players alternately choose one of the two sets T_0 or T_1 and build one row consisting of 2^n tiles from the chosen set. The first move is performed by Prover. Prover wins if after a finite number of moves there is a correctly tiled corridor, otherwise Spoiler wins. There are two possibilities for Spoiler to win: either Prover cannot make a move while the last constructed row is not in L^* or the game lasts forever.

We say that a tiling game $\langle \_, T_0, T_1, \_, \_, \_, \_\rangle$ is *well-formed* if for any play of the game and for any partial (i.e., non-complete) tiling constructed during this play, exactly one new row can be correctly constructed from tiles in T_0 and exactly one from tiles in T_1. In other words, every possible move in any play of the game is fully determined by the choice of the set of tiles, T_0 or T_1.

4.2.2 From Alternating Machines to Tiling Games

The first step of the lower-bound proof for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ is a reduction from alternating machines to tiling games. Actually, we have defined our version of tiling games in such a way that the proof of the following theorem becomes a routine exercise. The details can be found in the full version.

▶ **Theorem 13.** *For all alternating Turing machines M working in exponential space and all input words w there exists a well-formed tiling game of size polynomial in the sizes of M and w such that Prover has a winning strategy in the game if and only if w is accepted by M.*

The theorem above directly leads to the lower bound on the complexity of tiling games.

▶ **Corollary 14.** *The problem whether Prover has a winning strategy in a tiling game is 2-*ExpTime*-hard.*

4.2.3 From Tilling Games to $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$

Here we show the second step of the lower-bound proof for $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$, which is a reduction from tiling games to $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$. We are going to encode strategies for the existential player as trees. Every complete path in such an encoding corresponds to a correct tiling for G. The nodes on such a path, read from the root to the leaf, correspond to tiles in the tiling, read row by row from left to right. Intuitively, most nodes have just one child corresponding

to the tile placed directly to the right. Nodes corresponding to tiles in the last column should have one or two children, depending on whether it is the existential or universal player's turn. Formally the situation is a bit more complicated, because we are not able to prevent the tree from having additional branches with no meaning (for example, a node may have several children, each of which encodes the same right neighbor).

For a given number n we will be using unary predicates $B_0, \ldots, B_{n-1}$ for counting modulo 2^n; the predicate B_i will be responsible of the i-th bit of the corresponding number; B_0 is the least significant bit. This is a standard construction that can be found e.g. in [13] or [4], so we do not present the details here. In the following we will be using several macros that expand in an obvious way to formulas of length polynomial in n over predicates $B_0, \ldots, B_{n-1}$. Some of them are listed below.

$Nr(x) = 0$	the number encoded in the node x is 0
$Nr(x) = 2^n - 1$	the number encoded in the node x is $2^n - 1$
$Nr(x) = Nr(y) + 1$	the number in x is the successor of the number in y
$Nr(x) > Nr(y)$	the number in x is greater than the number in y
$\ldots$	$\ldots$

For example, the macro $Nr(x) > Nr(y)$ expands to

$$\bigvee_{i=0}^{n} \Big(B_i(x) \wedge \neg B_i(y) \wedge \bigwedge_{j=i+1}^{n} (B_j(x) \Leftrightarrow B_j(y)) \Big)$$

▶ **Theorem 15.** *For all well-formed tiling games G there exists a formula $\varphi \in \mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ of size polynomial in the size of G such that the existential player has a winning strategy in G if and only if φ is satisfiable over finite trees.*

Sketch of proof. Let $G = \langle C, T_0, T_1, n, \langle t_0, \ldots, t_n \rangle, \square, L \rangle$. We are going to define the formula φ as a conjunction of several smaller formulas responsible for different aspects of the encoding. Most of these aspects are routine; the most interesting is adjacency.

Numbering of nodes. We start by numbering nodes on paths in the underlying tree. These numbers encode the column numbers of the corresponding tiles. The following formulas express that the root is numbered 0, the number of any other node is the number of its father plus one modulo 2^n, and that all rows are complete (i.e., they have 2^n tiles).

$$\forall x \big(\neg(\exists y\ y{\downarrow}x) \Rightarrow Nr(x) = 0 \big)$$
$$\forall x \big(Nr(x) \neq 2^n - 1 \Rightarrow (\exists y\ x{\downarrow}y) \wedge \forall y\ (x{\downarrow}y \Rightarrow Nr(y) = Nr(x) + 1) \big)$$
$$\forall x \big(Nr(x) = 2^n - 1 \Rightarrow \forall y\ (x{\downarrow}y \Rightarrow Nr(y) = 0) \big)$$

Tiles and colors. Let $t_0, \ldots, t_m$ be an enumeration of all tiles occurring in the game, that is in $T_0 \cup T_1 \cup \{t_0, \ldots, t_n, \langle \square, \square, \square, \square \rangle\}$. The predicates $\mathit{tile}_1, \ldots, \mathit{tile}_m$ correspond to these tiles. For each color $c \in C$ we introduce four predicates $\pi^c_1, \pi^c_2, \pi^c_3, \pi^c_4$ that will be used to encode the four colors of a tile. The formulas below express that each node in the underlying tree corresponds to precisely one tile and is colored with the four colors of the tile. We assume here that $t_i = \langle c^1_i, c^2_i, c^3_i, c^4_i \rangle$. We also introduce predicates setT_0, setT_1 and setL corresponding to the sets T_0, T_1 and L, respectively.

$$\forall x\Big(\bigvee_{i=0}^{m}\big(tile_i(x)\wedge\bigwedge_{j\neq i}\neg tile_j(x)\big)\Big)$$

$$\forall x\bigwedge_{i=1}^{4}\Big(\bigvee_{c\in C}^{m}\big(\pi_i^c(x)\wedge\bigwedge_{c'\neq c}\neg\pi_i^{c'}(x)\big)\Big)$$

$$\bigwedge_{i=0}^{m}\Big(\forall x\big(tile_i(x)\Leftrightarrow\pi_1^{c_i^1}(x)\wedge\pi_2^{c_i^2}(x)\wedge\pi_3^{c_i^3}(x)\wedge\pi_4^{c_i^4}(x)\big)\Big)$$

$$\Big(\big(\bigvee_{t_i\in T_0}tile_i(x)\big)\Leftrightarrow setT_0(x)\Big)\wedge\Big(\big(\bigvee_{t_i\in T_1}tile_i(x)\big)\Leftrightarrow setT_1(x)\Big)\wedge\Big(\big(\bigvee_{t_i\in L}tile_i(x)\big)\leftrightarrow setL(x)\Big)$$

First and last row. The predicates *First* and *Last* are used to distinguish the first and the last row of a correct tiling. A node x corresponds to a tile in the first row if there is no other tile in the same column in previous rows. The last row is described dually. The first row is built from tiles $t_0,\ldots,t_n$ and padded out with white tiles, the last row is built only from tiles in L. The formulas expressing these properties are quite obvious and we omit them here, but they can be found in [2].

Existential and universal rows. Each element in each row is marked with predicate E or A depending on which player's turn it is. Each row is marked the same (each element has the same marking as its left neighbor, if it exists). The first row is existential and then the marking alternates between existential and universal.

Universal rows have two successors. Each non-first row is marked with predicate $move_0$ or $move_1$ depending on the set of tiles (T_0 or T_1) from which it is built. Universal non-last rows have two successors, marked respectively with $move_0$ and $move_1$.

Horizontal adjacency. This is simple. We first establish the white frame on the first and last tile in each row and then simply say that for each non-first tile in any row the left edge of the tile matches the right edge of the preceding tile.

$$\forall x\ \big(Nr(x)=0\Rightarrow\pi_1^{\square}(x)\wedge\pi_3^{\square}(x)\big)$$

$$\forall x\ \big(Nr(x)=2^n{-}1\Rightarrow\pi_2^{\square}(x)\wedge\pi_4^{\square}(x)\big)$$

$$\bigwedge_{c\in C}\Big(\forall x\big((Nr(x)\neq 0)\wedge\pi_1^c(x)\Rightarrow\exists y\ (y{\downarrow}x\wedge\pi_2^c(y))\big)\Big)$$

$$\bigwedge_{c\in C}\Big(\forall x\big((Nr(x)\neq 0)\wedge\pi_3^c(x)\Rightarrow\exists y\ (y{\downarrow}x\wedge\pi_4^c(y))\big)\Big)$$

Vertical adjacency. This is the tricky part of the encoding. The difficulty comes from the fact that we cannot number rows of the tiling and we have no means to say that two tiles occur in consecutive rows. Using predicates $B_0,\ldots,B_{n-1}$ we can number the 2^n columns, but the number of rows may be much higher and we cannot afford having enough predicates for numbering them. Therefore, we can say that two tiles occur in the same column (or in consecutive columns), but we cannot do the same with rows. On the other hand, when we add a new tile to a row, we have to make sure that the upper edge of the new tile matches

the lower edge of the tile directly above, so we have to read the colors of the tile directly above. For non-white colors this can be done by observing that each occurrence of a color in an upper edge of a row must be matched with another occurrence of the same color in a lower edge of the previous row in the same column, therefore the number of occurrences of each color in each column should be even. Hence the color directly above the upper edge of the current row is the only non-white color that occurs an odd number of times in the current column in preceding rows. In the case of white color we have to take into consideration the upper white edge of the constructed tiling, so the color directly above is white if it occurs an even number of times in the current column in preceding rows.

$$\forall x\big(\pi_1^{\square}(x) \Rightarrow \exists^{=0,2} y \;\; y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_1^{\square}(y) \vee \pi_3^{\square}(y))\big)$$
$$\forall x\big(\pi_2^{\square}(x) \Rightarrow \exists^{=0,2} y \;\; y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_2^{\square}(y) \vee \pi_4^{\square}(y))\big)$$
$$\bigwedge_{c\in C\setminus\{\square\}} \Big(\forall x\big(\pi_1^{c}(x) \Rightarrow \exists^{=1,2} y \; y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_1^{c}(y) \vee \pi_3^{c}(y))\big)\Big)$$
$$\bigwedge_{c\in C\setminus\{\square\}} \Big(\forall x\big(\pi_2^{c}(x) \Rightarrow \exists^{=1,2} y \; y{\downarrow^+}x \wedge Nr(y){=}Nr(x) \wedge (\pi_2^{c}(y) \vee \pi_4^{c}(y))\big)\Big)$$

Correctness of the constructed formula. Let φ be the conjunction of all formulas mentioned above. It is not difficult to see that the size of φ is polynomial in the size of G: the longest part of φ is the encoding of tiles and colors, which is proportional in length to the sum of squared numbers of tiles and colors.

Assume that φ has a finite model $\mathfrak{M}$. The formula *Tiles and colors* guarantees that each node in $\mathfrak{M}$ directly encodes precisely one tile. The formula *Numbering of nodes* guarantees that nodes on each root-to-leaf path are correctly numbered modulo 2^n, with the root numbered 0 and the leaf numbered $2^n - 1$. Thus each segment of length 2^n of such a path, consisting of nodes numbered from 0 to $2^n - 1$, corresponds to one row of tiles. The *Horizontal adjacency* formula guarantees that (horizontally) adjacent tiles in such a row have matching colors. Similarly, *Vertical adjacency* formula guarantees that tiles occurring on the same position in two consecutive rows (that is, vertically adjacent tiles) have matching colors. The *First and last row* formula guarantees that the first row is initial and the last row is built from tiles in L. Therefore each complete path in $\mathfrak{M}$ encodes a correctly tiled corridor. By the *Existential and universal rows* formula all rows encoded in any such path are alternately marked as existential and universal, starting with an existential one. Finally, the *Universal rows have two successors* formula guarantees that each encoding of a universal row in $\mathfrak{M}$ is followed by encodings of two existential rows, one build from tiles in T_0 and one from tiles in T_1. Thus (here we use the assumption that the game is well-formed) $\mathfrak{M}$ covers both possible moves of the universal player. Now the strategy of the existential player is to follow the path in $\mathfrak{M}$ corresponding to the partial tiling as it is constructed during the game. She starts in the root of $\mathfrak{M}$. Then, every time when it is her turn, the existential player reads from $\mathfrak{M}$ any successor row of her current position and replies with this row, moving down the tree to the position of the successor row. Every time when it is the universal player's turn, his both possible moves are encoded as successor rows of the current position of the existential player, so she can always follow the branch chosen by the universal player. Since $\mathfrak{M}$ is finite, each play stops after finitely many rounds with the constructed tiling corresponding to a path in $\mathfrak{M}$. Therefore in each play we obtain a correctly tiled corridor and the existential player wins.

For the other direction, assume that the existential player has a winning strategy. We construct a model $\mathfrak{M}$ of φ inductively, level by level, as follows. We start with 2^n nodes, number them from 0 to $2^n - 1$ using predicates $B_0, \ldots, B_{n-1}$ and connect consecutive nodes with predicate $\downarrow$. Then we label nodes numbered 0 to n with predicates $\mathit{tile}_0, \ldots, \mathit{tile}_n$, respectively, and nodes numbered $n+1$ to $2^n - 1$ with $\langle \square, \square, \square, \square \rangle$. Then (and later, always after adding new labels of the form tile_i) we add appropriate labels $\pi_j^c, \mathit{setT}_0, \mathit{setT}_1$ and setL to make the formula *Tiles and colors* true. Similarly, we add labels $\downarrow^+$ to all pairs of nodes that are connected by the transitive closure of $\downarrow$. We also label all these nodes with predicates *First* and E. This way we obtain the encoding of the initial row in the game. Next we inductively, level by level, construct the remaining parts of $\mathfrak{M}$.

Assume that $\mathfrak{M}$ is constructed up to some level k, with all leaves labeled E, and that each path constructed so far encodes a partial tiling in some play after k rounds, with existential player's turn. We extend $\mathfrak{M}$ leaf by leaf. Let us consider one such leaf ℓ, it encodes the last tile in some row. If all tiles in this row are in the set L, we label all nodes in this row with predicate *Last* and finish the construction. Otherwise let r be the next row given by the winning strategy of the existential player in the partial play encoded by the path from the root to ℓ. We take fresh 2^n nodes and extend the path from root to ℓ with the encoding of a row r, as above, but this time labeling all new nodes with predicate A indicating the universal player's move. We also label all new nodes with predicate move_0 or move_1, depending on whether r is constructed from tiles in T_0 or T_1, respectively. Again, if the row is final, it is marked with predicate *Last*, otherwise we take twice 2^n fresh nodes and connect to the current leaf two segments of length 2^n that encode two possible moves of the universal player. We mark each node in both segments with predicate E and thus finish the construction of level $k+1$ at node ℓ. The construction is repeated for all leafs at level k. Since during the construction we follow the winning strategy, no encoded play can last forever and the construction ends after finitely many iterations. By inspection of all conjuncts one can check that the constructed tree is a model of φ. ◀

As a corollary of the theorem above and Corollary 14 we get the main theorem of this section.

▶ **Theorem 16.** *The satisfiability problem for* $\mathrm{FO}^2_{\mathrm{MOD}}[\downarrow, \downarrow^+]$ *interpreted over finite trees is 2-*ExpTime*-hard.*

5 Conclusions and future work

We have shown that the satisfiability problem for two-variable logic extended with modulo counting quantifiers and interpreted over finite trees is 2-ExpTime-complete. The upper bound is based on the small-model property of the logic; for the lower bound we have developed a version of tiling games for AExpSpace computations.

There are several possible directions for future work. One of them is studying restrictions or extensions of the logic presented here. Natural candidates for restrictions include *guarded fragment* of the logic or *unary alphabet restriction* as in [4]; natural extensions include arbitrary uninterpreted binary symbols as in [3]. Another possibility is investigation of the (finite) satisfiability problem for $\mathrm{FO}^2_{\mathrm{MOD}}$ on arbitrary structures – we even do not know whether this problem is decidable. Yet another direction is to study the expressive power of the logic and to find an expressively equivalent extension of CTL.

References

1 Augustin Baziramwabo, Pierre McKenzie, and Denis Thérien. Modular temporal logic. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 344–351. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782629.

2 Bartosz Bednarczyk and Witold Charatonik. Modulo counting on words and trees. *CoRR*, http://arxiv.org/abs/1710.05582, 2017. URL: http://arxiv.org/abs/1710.05582.

3 Bartosz Bednarczyk, Witold Charatonik, and Emanuel Kieronski. Extending two-variable logic on trees. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.11.

4 Saguy Benaim, Michael Benedikt, Witold Charatonik, Emanuel Kieroński, Rastislav Lenhardt, Filip Mazowiecki, and James Worrell. Complexity of two-variable logic on finite trees. *ACM Transactions on Computational Logic*, 17(4):32:1–32:38, 2017. Extended abstract in ICALP 2013. URL: http://dl.acm.org/citation.cfm?id=2996796.

5 Michael Benedikt and Luc Segoufin. Regular tree languages definable in FO and in fo_{mod}. *ACM Trans. Comput. Log.*, 11(1):4:1–4:32, 2009. doi:10.1145/1614431.1614435.

6 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICDT.2017.8.

7 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and a linear order. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:8)2016.

8 Bogdan S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986. doi:10.1016/0022-0000(86)90036-X.

9 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002. doi:10.1006/inco.2001.2953.

10 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 306–317. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614957.

11 Frederik Harwath and Nicole Schweikardt. On the locality of arb-invariant first-order formulas with modulo counting quantifiers. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:8)2016.

12 Lucas Heimberg, Dietrich Kuske, and Nicole Schweikardt. Hanf normal form for first-order logic with unary counting quantifiers. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 277–286. ACM, 2016. doi:10.1145/2933575.2934571.

13 Emanuel Kieronski. On the complexity of the two-variable guarded fragment with transitive guards. *Inf. Comput.*, 204(11):1663–1703, 2006. doi:10.1016/j.ic.2006.08.001.

14 Andreas Krebs and A. V. Sreejith. Non-definability of languages by generalized first-order formulas over (n, +). In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 451–460. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.55.

15 Kamal Lodaya and A. V. Sreejith. Two-variable first order logic with counting quantifiers: Complexity results. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2017. doi:10.1007/978-3-319-62809-7_19.

16 Maarten Marx and Maarten de Rijke. Semantic characterization of navigational XPath. In *First Twente Data Management Workshop (TDM 2004) on XML Databases and Information Retrieval*, pages 73–79, 2004.

17 Juha Nurmonen. Counting modulo quantifiers on finite structures. *Inf. Comput.*, 160(1-2):62–87, 2000. doi:10.1006/inco.1999.2842.

18 Andreas Potthoff. Modulo-counting quantifiers over finite trees. *Theor. Comput. Sci.*, 126(1):97–112, 1994. doi:10.1016/0304-3975(94)90270-4.

19 Amitabha Roy and Howard Straubing. Definability of languages by generalized first-order formulas over n^+. *SIAM J. Comput.*, 37(2):502–521, 2007. doi:10.1137/060658035.

20 Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

21 A. V. Sreejith. Expressive completeness for LTL with modulo counting and group quantifiers. *Electr. Notes Theor. Comput. Sci.*, 278:201–214, 2011. doi:10.1016/j.entcs.2011.10.016.

22 Howard Straubing and Denis Thérien. Modular quantifiers. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*., volume 2 of *Texts in Logic and Games*, pages 613–628. Amsterdam University Press, 2008.

23 Howard Straubing, Denis Thérien, and Wolfgang Thomas. Regular languages defined with generalized quanifiers. *Inf. Comput.*, 118(2):289–301, 1995. doi:10.1006/inco.1995.1067.

Probabilistic Disclosure: Maximisation vs. Minimisation*

Béatrice Bérard[1], Serge Haddad[†2], and Engel Lefaucheux[3]

1 **Sorbonne Universités, UPMC Univ. Paris 06, CNRS UMR 7606, LIP6, Paris, and LSV, ENS Paris-Saclay & CNRS & Inria, Université Paris-Saclay, France**
beatrice.berard@lip6.fr

2 **LSV, ENS Paris-Saclay & CNRS & Inria, Université Paris-Saclay, France**
serge.haddad@lsv.fr

3 **Inria, Campus Universitaire de Beaulieu, Rennes, France and LSV, ENS Paris-Saclay & CNRS & Inria, Université Paris-Saclay, France**
engel.lefaucheux@inria.fr

Abstract

We consider opacity questions where an observation function provides to an external attacker a view of the states along executions and secret executions are those visiting some state from a fixed subset. Disclosure occurs when the observer can deduce from a finite observation that the execution is secret, the ε-disclosure variant corresponding to the execution being secret with probability greater than $1-\varepsilon$. In a probabilistic and non deterministic setting, where an internal agent can choose between actions, there are two points of view, depending on the status of this agent: the successive choices can either help the attacker trying to disclose the secret, if the system has been corrupted, or they can prevent disclosure as much as possible if these choices are part of the system design. In the former situation, corresponding to a worst case, the disclosure value is the supremum over the strategies of the probability to disclose the secret (maximisation), whereas in the latter case, the disclosure is the infimum (minimisation). We address quantitative problems (comparing the optimal value with a threshold) and qualitative ones (when the threshold is zero or one) related to both forms of disclosure for a fixed or finite horizon. For all problems, we characterise their decidability status and their complexity. We discover a surprising asymmetry: on the one hand optimal strategies may be chosen among deterministic ones in maximisation problems, while it is not the case for minimisation. On the other hand, for the questions addressed here, more minimisation problems than maximisation ones are decidable.

1998 ACM Subject Classification D.2.4 Software, Program Verification

Keywords and phrases Partially observed systems, Opacity, Markov chain, Markov decision process

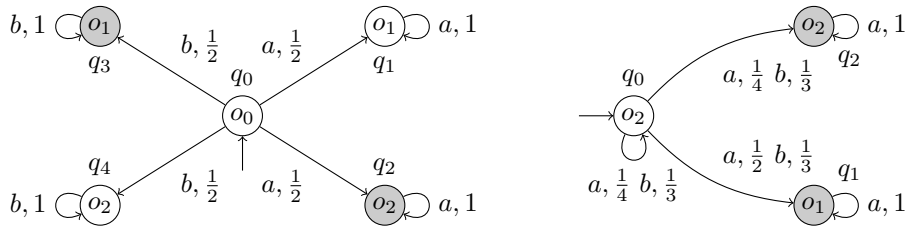
Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.13

1 Introduction

Opacity. Opacity of an information system is a key security property: an external user should not, by observing an execution of a system, acquire the guarantee that it is a secret one. This property was first formalised for labelled transition systems [7], by specifying

* Full proofs can found in research report https://hal.inria.fr/hal-01618955.

† The work of S. Haddad has been supported by ERC project EQualIS (FP7-308087).



(a) $Disc = 0$ or $\frac{1}{2}$. **(b)** $Disc = \frac{2}{3}$.

Figure 1 When strategies help (or not) to disclose a secret.

a subset of secret paths and requiring that, for any secret path, there is a non-secret one with the same observation. The *disclosure set* of a system is then the set of (secret) paths violating opacity.

Opacity raises challenging research issues like (1) formal specification in various frameworks [13, 7], (2) the design of mechanisms to ensure opacity while preserving functionality and performance [2], and (3) the verification of opacity properties [14, 7].

Attacks against opacity. In order to quantify the size of the leak, various measures for the disclosure set, called probabilistic disclosure, were introduced in [17, 3, 5, 4]. For probabilistic and non deterministic systems like Markov Decision Processes (MDP), where an internal agent can choose between several actions, the disclosure can be either maximised or minimised depending on the status of the agent. In previous works, the situation considered corresponded to maximisation, where the system has been corrupted (*e.g.*, by a virus), and the internal agent cooperates with the attacker to disclose the secret. In [3], the set of secret paths is specified by some deterministic automaton and the attacker does not know which strategy is applied, hence the set of executions leaking the secret is fixed by the structure of the system and does not vary according to the strategies. This is illustrated in Figure 1a with an MDP where actions a and b are possible from state q_0. Action a (resp. b) has a uniform distribution over states q_1 and q_2 (resp. q_3 and q_4). States q_1 and q_3 produce observation o_1, while q_2 and q_4 produce o_2. The secret paths are those in $q_0q_2^\omega \cup q_0q_3^\omega$ (reaching either q_2 or q_3, in grey). If the observer is not aware of the strategy and thus has to consider all possible paths whatever the strategy, the non secret paths are $q_0q_1^\omega$ and $q_0q_4^\omega$, hence there is no disclosing path, leading to a null disclosure (as in [3]). On the other hand, if the observer is aware of the strategy, assuming that initially a is chosen produces a maximal disclosure of $\frac{1}{2}$. This example also shows that deterministic strategies are not sufficient to achieve minimisation: value 0 for the disclosure can only be obtained by randomised strategies, choosing a with probability p and b with probability $1-p$ (for $0 < p < 1$) in q_0.

In contrast, Figure 1b represents an MDP where both actions a and b have the same support in state q_0, hence choosing a or b does not change the states that can be reached. Under a strategy which always plays a, the disclosure is equal to $\frac{2}{3}$ which is the probability of reaching q_1.

Given some $\varepsilon > 0$, a path could alternatively be considered as disclosing the secret, if the measure of the set of paths with same observation that are not secret is less than ε. This notion is called ε-disclosure. For instance in Figure 1a with $\varepsilon < \frac{1}{2}$ in order to achieve a minimal ε-disclosure of 0 the strategy must select p between ε and $1-\varepsilon$. However in Figure 1b for every $\varepsilon > 0$ the ε-disclosure is equal to 1 as the probability to be in a secret state converges to 1 on every path.

Table 1 Complexity results for maximisation and minimisation of disclosure.

Disclosure	General	Limit-sure	Almost-sure
Maximisation finite horizon	undecidable	undecidable	EXPTIME-c
Minimisation finite horizon	PSPACE-hard $\leq$ Min $\leq$ EXPTIME		
Maximisation fixed horizon	PSPACE-c.	PTIME	PTIME
Minimisation fixed horizon	PSPACE-c	PSPACE-c	PSPACE-c

This figure also illustrates the drastic restriction used in [4] where no edge in an MDP can be blocked by a strategy. With this restricted power of the internal component, the authors can assume that the observer knows the strategy, which is an important requirement since the security of a system should not be based on hiding its design. The model used in [4] is in fact a restricted case of Interval Markov Decision Processes (IMDPs) so that after some transformation, the problem boils down to IMDP model checking [10]. The general decidability status of the disclosure problem is left open.

Contributions. Here we focus on several problems in MDPs under partial observation that cannot be formalised as problems for classical POMDPs (Partially Observable MDPs). The notion of disclosure is defined with respect to a fixed subset Sec of states: A (finite or infinite) path is secret if it has visited some state of Sec. Other variants of secret path specifications have been proposed with deterministic finite automata accepting finite or infinite paths. The former case can be easily translated in our setting while we believe that the latter one is debatable: a system is not really vulnerable if the attacker can only know the secret at infinite horizon!

Once a strategy is fixed, the behaviour of the system is described by a possibly infinite partially observable Markov chain, so we start in Section 2 by establishing several results on the semantical aspects of disclosure in Markov chains. In addition, we prove undecidability for the positive ε-disclosure problem (deciding if ε-disclosure is positive) within finite horizon. We then consider two different settings depending on the status of the strategies. Like in previous work, maximisation of disclosure corresponds to the internal agent cooperating with the attacker to disclose a secret. Dually, minimisation is interesting to study during the system design process, in order to optimise the choices of the internal agent to defend the system. We address various problems in these settings, for a finite horizon but also for a fixed horizon (given in unary representation), corresponding to real-time constraints requiring the number of steps to be fixed in advance. The quantitative decision problem asks whether the disclosure is above or below some threshold, while qualitative problems consider extremal values (0 or 1) of the disclosure. We prove that observation-based strategies (*i.e.*, which only depend on the sequence of observations and the current state) are dominant in both cases.

The main complexity results for decision problems are gathered in Table 1. For the maximisation objective (Section 3), we show that deterministic strategies are dominant. We answer negatively to the decidability issues left open in [4], proving that both the quantitative problem and the limit-sure problem (asking whether the supremum over all strategies is 1) are undecidable for a finite horizon. Then, we show that the almost-sure problem (asking whether there is a strategy producing a value 1 for disclosure) is EXPTIME-complete. For minimisation (Section 4), we introduce families of randomised strategies, necessary to asymptotically reach minimal disclosure, even within fixed horizon. For finite horizon, we show that the computation and decision problems belong to EXPTIME. Hence surprisingly, although the problem seems more difficult due to the necessity of randomised strategies, the

disclosure problem for minimisation is decidable whereas it is not for maximisation. Section 5 is devoted to the fixed horizon problems. For maximisation, we prove that the disclosure value can be computed in PSPACE (while its associated strategy can be computed in EXPTIME) and also establish that the corresponding decision problem is PSPACE-complete. The almost-sure and limit-sure decision problem however are easier and can be solved in PTIME. Refining the techniques to take randomised strategies into account, we obtain PSPACE-completeness of the various decision problems for minimisation.

2 Specification

We denote by $\mathbb{N}$ the set of natural numbers. For a finite alphabet Σ, we denote by Σ^* (resp. Σ^ω) the set of finite (resp. infinite) words over Σ, with $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ and ε the empty word. The length of a word w is denoted by $|w| \in \mathbb{N} \cup \{\infty\}$ and for $n \in \mathbb{N}$, Σ^n is the set of words of length n. A word $u \in \Sigma^*$ is a prefix of $v \in \Sigma^\infty$, written $u \leq v$, if $v = uw$ for some $w \in \Sigma^\infty$. The prefix is strict if $w \neq \varepsilon$. Given a countable set Z, a distribution on Z is a mapping $\mu : Z \to [0,1]$ such that $\sum_{z \in Z} \mu(z) = 1$. The support of μ is $Supp(\mu) = \{z \in Z \mid \mu(z) > 0\}$. If $Supp(\mu) = \{z\}$ is a single element, μ is a Dirac distribution on z written $\mathbf{1}_z$. We denote by $\mathsf{Dist}(S)$ the set of distributions on S.

2.1 Opacity for Markov chains

For the purpose of opacity questions, the models are equipped with a labelling function on states, called *observation function*, describing what an external observer can see. We first define observable Markov chains (MCs for short).

▶ **Definition 1** (Markov chains)**.** An observable Markov chain (MC) over alphabet Σ is a tuple $\mathcal{M} = (S, p, \mathsf{O})$ where S is a countable set of states, $p : S \to \mathsf{Dist}(S)$ is the transition function, and $\mathsf{O} : S \to \Sigma \cup \{\varepsilon\}$ is the observation function.

We write $p(s'|s)$ instead of $p(s)(s')$ to emphasise the probability of going to state s' conditionned by being in state s. Given a distribution μ_0 on S, we denote by $\mathcal{M}(\mu_0)$ the chain with initial distribution μ_0. An infinite path of $\mathcal{M}(\mu_0)$ is a sequence of states $\rho = s_0 s_1 \ldots \in S^\omega$ such that $\mu_0(s_0) > 0$ and for each $i \geq 0$, $p(s_{i+1}|s_i) > 0$. A finite path of length n is a prefix $\rho = s_0 s_1 \ldots s_n$ of an infinite path, ending in state $\mathsf{last}(\rho) = s_n$. We denote by $\mathsf{Path}(\mathcal{M}(\mu_0))$ (resp. $\mathsf{FPath}(\mathcal{M}(\mu_0))$) the set of infinite (finite) paths of $\mathcal{M}(\mu_0)$. The observation of path $\rho = s_0 s_1 \ldots$ is the word $\mathsf{O}(\rho) = \mathsf{O}(s_0)\mathsf{O}(s_1)... \in \Sigma^\infty$. For a set R of paths, $\mathsf{O}(R) = \{\mathsf{O}(\rho) \mid \rho \in R\}$ and for a set W of observations, $\mathsf{O}^{-1}(W) = \{\rho \mid \mathsf{O}(\rho) \in W\}$. The observation function is called *non erasing* if $\mathsf{O}(S) \subseteq \Sigma$ (all states are visible).

A probability measure $\mathbf{P}_{\mathcal{M}(\mu_0)}$ is defined on $\mathsf{Path}(\mathcal{M}(\mu_0))$, where the measurable sets are generated by the cylinders $\mathsf{Cyl}(\rho)$, for $\rho \in \mathsf{FPath}(\mathcal{M}(\mu_0))$, containing the infinite paths having ρ as prefix. Then $\mathbf{P}_{\mathcal{M}(\mu_0)}$ is inductively defined by: $\mathbf{P}_{\mathcal{M}(\mu_0)}(s) = \mu_0(s)$ for $s \in S$ and for $\rho' = \rho s'$, with $\mathsf{last}(\rho) = s$, $\mathbf{P}_{\mathcal{M}(\mu_0)}(\mathsf{Cyl}(\rho')) = \mathbf{P}_{\mathcal{M}(\mu_0)}(\mathsf{Cyl}(\rho))p(s'|s)$. We sometimes write $\mathbf{P}_{\mathcal{M}(\mu_0)}(\rho)$ instead of $\mathbf{P}_{\mathcal{M}(\mu_0)}(\mathsf{Cyl}(\rho))$ for $\rho \in \mathsf{FPath}(\mathcal{M})$ and for $w \in \Sigma^*$, $\mathbf{P}_{\mathcal{M}(\mu_0)}(w)$ instead of $\mathbf{P}_{\mathcal{M}(\mu_0)}(\cup_{\rho \in \mathsf{O}^{-1}(w)} \mathsf{Cyl}(\rho))$.

We consider here the particular case where the secret is given by a subset of states $Sec \subseteq S$ of the model: a (finite of infinite) path $s_0 s_1 \ldots$ is secret if $s_i \in Sec$ for some i. We first define a probabilistic version of disclosure w.r.t. some $\varepsilon > 0$ to answer the question: Is there non-zero probability of observing some w that has probability more than $1 - \varepsilon$ of coming from a secret path?

▶ **Definition 2** (ε-Disclosure)**.** Given an MC $\mathcal{M} = (S, p, \mathsf{O})$, an initial distribution μ_0, $Sec \subseteq S$ and an observation $w \in \Sigma^*$, the proportion of secret paths with observation w is:

$$\mathsf{Psec}_{\mathcal{M}(\mu_0)}(w) = \frac{\mathbf{P}_{\mathcal{M}(\mu_0)}(\{\rho \in \mathsf{O}^{-1}(w) \mid \rho \text{ is secret}\})}{\mathbf{P}_{\mathcal{M}(\mu_0)}(w)}.$$

For $\varepsilon > 0$, w is ε-min-disclosing if $\mathsf{Psec}_{\mathcal{M}(\mu_0)}(w) > 1 - \varepsilon$ and no prefix of w satisfies this inequality. Writing $D^{\varepsilon}_{\min}$ for the set of ε-min-disclosing observations, the ε-disclosure is defined by $Disc^{\varepsilon}(\mathcal{M}(\mu_0)) = \sum_{w \in D^{\varepsilon}_{\min}} \mathbf{P}_{\mathcal{M}(\mu_0)}(w)$. The positive ε-disclosure problem consists in deciding if $Disc^{\varepsilon}(\mathcal{M}(\mu_0)) > 0$.

While being the most realistic notion of probabilistic disclosure, unfortunately the problem is already undecidable for Markov chains:

▶ **Theorem 3.** *The positive ε-disclosure problem is undecidable for MCs.*

Like in further proofs, we use a reduction from a problem on Probabilistic Automata (PA). Recall that a PA is a tuple $\mathcal{A} = (Q, q_0, \mathsf{Act}, T, F)$ where Q is a finite set of states with $q_0 \in Q$ the initial state, Act is a finite set of actions, $T : Q \times \mathsf{Act} \to \mathsf{Dist}(Q)$ is the transition function and $F \subseteq Q$ is the set of final states.

For a finite path $\rho = q_0 \xrightarrow{(} a_1) q_1 \dots \xrightarrow{(} a_n) q_n$ of $\mathcal{A}$, the word $a_1 \dots a_n \in \mathsf{Act}^*$ is called the *trace* of ρ and denoted by $tr(\rho)$. Writing $\mathsf{FPath}(w, q) = \{\rho \in \mathsf{FPath} \mid tr(\rho) = w \text{ and } \mathsf{last}(\rho) = q\}$ for $w \in \mathsf{Act}^*$ and $q \in Q$, we define $\mathbf{P}^q_{\mathcal{A}}(w) = \mathbf{P}_{\mathcal{A}}(\cup_{\rho \in \mathsf{FPath}(w,q)} \mathsf{Cyl}_\rho)$, $\mathbf{P}^F_{\mathcal{A}}(w) = \sum_{q \in F} \mathbf{P}^q_{\mathcal{A}}(w)$ and $val(\mathcal{A}) = sup_{w \in \mathsf{Act}^*} \mathbf{P}^F_{\mathcal{A}}(w)$.

Given a threshold $\theta \in [0, 1[$, we set $\mathcal{L}_{>\theta}(\mathcal{A}) = \{w \in \mathsf{Act}^* \mid \mathbf{P}^F_{\mathcal{A}}(w) > \theta\}$. The strict emptiness problem for $\mathcal{A}$, asking whether this set is empty or not, is known to be undecidable for $\theta > 0$ [16]. The value 1 problem, asking whether $val(\mathcal{A}) = 1$ is undecidable as well [11].

Sketch of Proof. Given a PA $\mathcal{A}$, we build a Markov chain $\mathcal{M}_{\mathcal{A}}$ with initial distribution μ_0 and secret Sec such that for any ε, $0 < \varepsilon < 1$, $\mathcal{L}_{>1-\varepsilon}(\mathcal{A})$ is not empty iff $Disc^{\varepsilon}(\mathcal{M}_{\mathcal{A}}(\mu_0)) > 0$. ◀

This leads us to return to the simpler case where the disclosure is the probability of the set of paths leaking the secret, *i.e.*, such that *all* paths with the same observation are secret. The ω-disclosure (corresponding to measures in [3, 5, 4]) was defined for a Markov chain $\mathcal{M} = (S, p, \mathsf{O})$ with initial distribution μ_0 by considering a measurable set of secret paths $\mathsf{SPath} \subseteq \mathsf{Path}(\mathcal{M}(\mu_0))$. Here, as mentioned above, SPath is $\mathsf{Reach}(Sec)$, the set of infinite paths visiting a state from Sec, and an infinite observation $w \in \Sigma^\omega$ discloses the secret if all paths $\rho \in \mathsf{O}^{-1}(w)$ are secret. Setting $\overline{\mathsf{SPath}} = \mathsf{Path}(\mathcal{M}(\mu_0)) \setminus \mathsf{SPath}$, we define:

▶ **Definition 4** (ω-Disclosure)**.** For an MC $\mathcal{M} = (S, p, \mathsf{O})$, an initial distribution μ_0 and a subset $Sec \subseteq S$, with $\mathsf{SPath} = \mathsf{Reach}(Sec)$, the ω-disclosure is defined by:
$Disc_\omega(\mathcal{M}(\mu_0)) = \mathbf{P}_{\mathcal{M}(\mu_0)}(\mathsf{SPath} \setminus \mathsf{O}^{-1}(\mathsf{O}(\overline{\mathsf{SPath}})))$.

To obtain measures directly related to the finite observation of a possible attacker, we assume that $\mathcal{M} = (S, p, \mathsf{O})$ is *convergent*: each infinite path ρ has an infinite observation $\mathsf{O}(\rho) \in \Sigma^\omega$. Two measures can then be defined, when considering a fixed or finite horizon. In the former case, we consider a non-erasing function O to obtain real-time observations.

▶ **Definition 5** (Disclosure of MCs)**.** Let $\mathcal{M} = (S, p, \mathsf{O})$ be an MC, μ_0 an initial distribution and $Sec \subseteq S$. A finite observation $w \in \Sigma^*$ discloses the secret if all paths $\rho \in \mathsf{O}^{-1}(w)$ are secret. It is min-disclosing if it discloses the secret and no strict prefix of w does.

***n*-disclosure:** When O is non-erasing, we denote by D_n, for $n \in \mathbb{N}$, the set of disclosing observations of length n. The n-disclosure is $Disc_n(\mathcal{M}(\mu_0)) = \sum_{w \in D_n} \mathbf{P}_{\mathcal{M}(\mu_0)}(w)$;

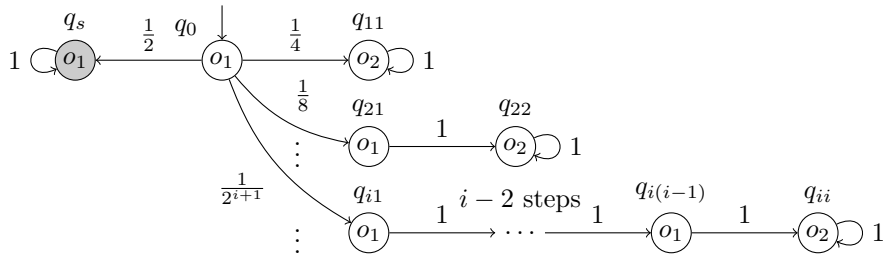


Figure 2 An infinitely branching MC with $Sec = \{q_s\}$, $Disc_\omega = \frac{1}{2}$ and $Disc = 0$.

Disclosure: Writing $D_{\min}$ for the set of min-disclosing observations, the disclosure (w.r.t. finite horizon) is defined by $Disc(\mathcal{M}(\mu_0)) = \sum_{w \in D_{\min}} \mathbf{P}_{\mathcal{M}(\mu_0)}(w)$.

Note that if D is the set of disclosing observations, and $\mathcal{V}(\mu_0) = \cup_{w \in D} \cup_{\rho \in \mathsf{O}^{-1}(w)} \mathsf{Cyl}(\rho)$ the set of paths disclosing the secret, then $Disc(\mathcal{M}(\mu_0))$ is also equal to $\mathbf{P}_{\mathcal{M}(\mu_0)}(\mathcal{V}(\mu_0))$.

▶ Remark. Without loss of generality, we can assume that once a secret state has been reached by an execution, all subsequent states remain secret. For this, a new Markov chain $\mathcal{M}' = (S', p', \mathsf{O}')$ is defined from $\mathcal{M}$ by: $S' = Sec \uplus ((S \setminus Sec) \times \{0,1\})$, where $(s,0)$ represents state s where the secret has not been visited while $(s,1)$ represents the opposite situation. The transitions are then duplicated accordingly: (1) $p'((s',i)|(s,i)) = p(s'|s)$ for all $s, s' \in S \setminus Sec$, and $i = 0,1$, (2) $p'((s',1)|s) = p(s'|s)$ for all $s \in Sec$, and $s' \in S \setminus Sec$, (3) $p'(s'|(s,i)) = p(s'|s)$ for all $s \in S \setminus Sec$, $i = 0,1$, and $s' \in Sec$, and (4) $p'(s'|s) = p(s'|s)$ for all $s, s' \in Sec$. The observation function is extended by $\mathsf{O}'((s,i)) = \mathsf{O}(s)$ for all $s \in S \setminus Sec$ and $i = 0,1$ and the new set of secrets is $Sec \uplus ((S \setminus Sec) \times \{1\})$. There is a one-to-one probability-preserving correspondence between the paths in $\mathcal{M}$ and those in $\mathcal{M}'$.

We show that disclosure and ω-disclosure may be different by consider the infinitely branching MC of Figure 2, with initial distribution $\mathbf{1}_{q_0}$, $Sec = \{q_s\}$ hence $\mathsf{SPath} = \{q_0 q_s^\omega\}$, $\mathsf{O}(\overline{\mathsf{SPath}}) = o_1^+ o_2^\omega$. Then $Disc_\omega = \frac{1}{2}$ but since no finite observation is disclosing, $Disc = 0$.

However, both notions coincide for convergent finitely branching MCs.

▶ **Lemma 6** (Comparison of Disclosure Notions). *Let $\mathcal{M} = (S, p, \mathsf{O})$ be a Markov chain, μ_0 an initial distribution and $Sec \subseteq S$. For $\mathsf{SPath} = \mathsf{Reach}(Sec)$, $Disc(\mathcal{M}(\mu_0)) \leq Disc_\omega(\mathcal{M}(\mu_0))$ and equality holds if $\mathcal{M}$ is convergent and finitely branching.*

2.2 Opacity for Markov Decision Processes

We now turn to MDPs that combine non determinism with probabilistic transitions.

▶ **Definition 7** (MDP). A Markov Decision Process (MDP) over alphabet Σ is a tuple $\mathsf{M} = (S, \mathsf{Act}, p, \mathsf{O})$ where S is a finite set of states, $\mathsf{Act} = \cup_{s \in S} A(s)$ where $A(s)$ is a finite non-empty set of actions for each state $s \in S$, $p : S \times \mathsf{Act} \to \mathsf{Dist}(S)$ is the (partial) transition function defined for (s,a) when $a \in A(s)$ and $\mathsf{O} : S \to \Sigma \cup \{\varepsilon\}$ is the observation function.

As before, we write $p(s'|s,a)$ instead of $p(s,a)(s')$. Given an initial distribution μ_0, an infinite path of M is a sequence $\rho = s_0 a_0 s_1 a_1 \ldots$ where $\mu_0(s_0) > 0$ and $p(s_{i+1}|s_i, a_i) > 0$, for $s_i \in S$, $a_i \in A(s_i)$, for all $i \geq 0$. Finite paths (ending in a state) and observation of a path are defined like for Markov chains, and we use similar notations for the various sets of paths. For decidability and complexity results, we assume that all probabilities occurring in the model (transition probabilities and initial distribution) are rationals.

Nondeterminism is resolved by strategies. Given a finite path ρ with $\mathsf{last}(\rho) = s$, a *decision rule* for ρ is a distribution on the possible actions in $A(s)$ chosen at this point. For such a decision rule δ, we write $p(s'|s,\delta) = \sum_{a\in A(s)} \delta(a)p(s'|s,a)$.

▶ **Definition 8** (Strategy). A strategy of MDP $\mathsf{M} = (S, \mathsf{Act}, p, \mathsf{O})$ with initial distribution μ_0 is a mapping $\sigma : \mathsf{FPath}(\mathsf{M}(\mu_0)) \to \mathsf{Dist}(\mathsf{Act})$ associating with ρ a decision rule $\sigma(\rho)$.

Given a strategy σ, a path $\rho = s_0a_0s_1a_1\dots$ of M is σ-compatible if for all i, $a_i \in Supp(\sigma(s_0a_0s_1a_1\dots s_i))$. A strategy σ is *deterministic* if $\sigma(\rho)$ is a Dirac distribution for each finite path ρ. In this case, we denote by $\sigma(\rho)$ the single action $a \in A(\mathsf{last}(\rho))$ such that $\sigma(\rho) = \mathbf{1}_a$. A strategy σ is *observation-based* if for any finite path ρ, $\sigma(\rho)$ only depends on the observation sequence $O(\rho)$ and the current state $\mathsf{last}(\rho)$, writing $\sigma(O(\rho), \mathsf{last}(\rho))$ for $\sigma(\rho)$.

Let σ be a strategy and ρ be a σ-compatible path. We define B^σ_ρ the *belief* of ρ w.r.t. σ about states corresponding to the last observation as follows:

$$B^\sigma_\rho = \{s \mid \exists \rho' \ \sigma\text{-compatible},\ O(\rho') = O(\rho) \wedge s = \mathsf{last}(\rho') \wedge O(s) \neq \varepsilon\}$$

A strategy σ is *belief-based* if for all ρ, $\sigma(\rho)$ only depends on its belief B^σ_ρ and its current state $\mathsf{last}(\rho)$. Observe that a belief-based strategy is observation-based since B^σ_ρ only depends on $w = \mathsf{O}(\rho)$. So we also write B^σ_w for B^σ_ρ. A strategy σ is *memoryless* if $\sigma(\rho)$ only depends on $\mathsf{last}(\rho)$ for all ρ.

A strategy σ on $\mathsf{M}(\mu_0)$ defines a (possibly infinite) Markov chain $\mathsf{M}_\sigma(\mu_0)$ with set of states $\mathsf{FPath}(\mathsf{M}_\sigma(\mu_0))$ (the finite σ-compatible paths), that can be equipped with the observation function associating $\mathsf{O}(\mathsf{last}(\rho))$ with the finite path ρ. The transition function p_σ is defined for $\rho \in \mathsf{FPath}(\mathsf{M}_\sigma(\mu_0))$ and $\rho' = \rho a s'$ by $p_\sigma(\rho'|\rho) = \sigma(\rho)(a)p(s'|s,a)$ and we denote by $\mathbf{P}_{\mathsf{M}_\sigma(\mu_0)}$ (or $\mathbf{P}_\sigma$ for short when there is no ambiguity) the associated probability measure. Writing $\mathcal{V}_\sigma(\mu_0)$ for the set of paths disclosing the secret in $\mathsf{M}_\sigma(\mu_0)$, we have $Disc(\mathsf{M}_\sigma(\mu_0)) = \mathbf{P}_{\mathsf{M}_\sigma(\mu_0)}(\mathcal{V}_\sigma(\mu_0))$.

Disclosure values for MDPs are defined according to the status of the strategies, by considering them as adversarial or cooperative with respect to the system (we only consider ε-disclosure for fixed horizon in view of the undecidability result of Theorem 3).

▶ **Definition 9** (Disclosure of an MDP). Given an MDP $\mathsf{M} = (S, \mathsf{Act}, p, \mathsf{O})$, an initial distribution μ_0 and a secret $Sec \subseteq S$, the maximal disclosure of Sec in M is $disc_{\max}(\mathsf{M}(\mu_0)) = \sup_\sigma disc(\mathsf{M}_\sigma(\mu_0))$ and the minimal disclosure is $disc_{\min}(\mathsf{M}(\mu_0)) = \inf_\sigma disc(\mathsf{M}_\sigma(\mu_0))$ for $disc \in \{Disc, Disc_n, Disc^\varepsilon_n\}$, $n \in \mathbb{N}$ and $0 < \varepsilon < 1$.

Note that the construction ensuring that once a secret state is visited, the path remains secret forever, extends naturally from Markov chains to MDPs. We consider only MDPs of this form in the sequel. We now show that for disclosure problems we can restrict strategies to observation-based ones.

▶ **Proposition 10** (Observation-based strategies). *Given an MDP, a secret and a strategy σ, there exists an observation-based strategy σ' with the same disclosure values.*

Erasing observations leads to technical and cumbersome developments. In order to avoid them in the design of procedures for the finite horizon case, we apply the preliminary transformation described in the next proposition. We precisely state the size of the transformed MDP in view of complexity results.

▶ **Proposition 11** (Avoiding erasing observations). *Given an MDP $\mathsf{M} = (S, \mathsf{Act}, p, \mathsf{O})$, an initial distribution μ_0 and a secret Sec, one can build in exponential time an MDP $\mathsf{M}' =$*

$(S', \mathsf{Act}', p', \mathsf{O}')$, an initial distribution μ'_0 and a secret Sec' where O' is non-erasing and for $disc \in \{Disc_{min}, Disc_{max}\}$ $disc(\mathsf{M}(\mu_0)) = disc(\mathsf{M}'(\mu'_0))$. In addition, the size of S', p' and μ'_0 is polynomial w.r.t. those of S', p' and μ'_0. The size of Act' is polynomial w.r.t. the size of Act and exponential w.r.t. the size of S.

We study the following problems over MDPs:

- **Computation problems.** The *value problem*: compute the disclosure and the *strategy problem*: compute an optimal strategy whenever it exists;
- **Quantitative decision problems.** The *disclosure problem*: Given M and a threshold $\theta \in [0, 1]$, is $disc(\mathsf{M}) \bowtie \theta$? with $\bowtie = \geq$ for maximisation and $\bowtie = \leq$ for minimisation, and the more demanding *strategy decision problem*: does there exist a strategy σ such that $disc(\mathsf{M}_\sigma) \bowtie \theta$?
- **Qualitative decision problems.** The *limit-sure disclosure problem*: the disclosure problem when $\theta = 1$ for maximisation and $\theta = 0$ for minimisation and the *almost-sure disclosure problem*: the strategy decision problem with the same restrictions.

For the complexity results regarding a fixed horizon n, we will assume that n is written in unary representation or bounded by a polynomial in the size of the model where the polynomial is independent of the model as done in classical studies (see for instance [15]).

3 Maximisation with finite horizon

While strategies may be randomised, this additional power is not necessary for maximisation:

▶ **Proposition 12** (Dominance of deterministic strategies)**.** *Given an MDP, a secret and an observation-based strategy σ there exists a deterministic observation-based strategy σ' with greater or equal disclosure of the secret.*

Sketch of Proof. The Lemma 1 of [8] (or alternatively [12]) does not directly give the result as, contrary to the objectives used in their paper, disclosure depends on the strategy. However, as a disclosing path for a randomised strategy is also a disclosing path for a deterministic strategy that does not introduce new paths, we can use parts of their proof to show our result. ◀

An edge can be completely blocked by some strategy, modifying the set of paths that disclose the secret. This was illustrated in Figure 1a, where choosing action a in state q_0 removes the edges to q_3 and q_4. This situation was excluded in the disclosure computation from [4], where the general problem was left open for Interval Markov Chains (IMCs). We answer negatively by proving undecidability of the disclosure problem, hence the disclosure cannot be computed in general. Undecidability also holds for limit-sure disclosure.

Writing $\mathbb{I}$ for the set of intervals in $[0, 1]$, an IMC (with observation) is a tuple $\mathsf{M} = (S, s_{init}, I, \mathsf{O})$ where S is the set of states, s_{init} is the initial state, $I : S \to \mathbb{I}^S$ associates with any state $s \in S$ a mapping from S into $\mathbb{I}$, and $\mathsf{O} : S \to \Sigma \cup \{\varepsilon\}$ is the observation function. An IMC can be transformed into an (exponentially larger) MDP where actions are the basic feasible solutions of the linear program specified by the constraints associated with intervals [18]. Thus undecidability results for IMCs also hold for MDPs.

▶ **Theorem 13** (Undecidability of maximal finite horizon disclosure)**.** *The maximal finite horizon disclosure problem is undecidable for MDPs, even when the secret is reached with probability 1 and for a non-erasing observation function.*
The maximal finite horizon disclosure problem when restricted to finite-memory strategies is also undecidable (with the same additional assumptions).

Sketch of Proof. Starting from a PA $\mathcal{A}$, we build an IMC $\mathsf{M}_{\mathcal{A}} = (S, s_0, I, \mathsf{O})$ such that there exists a word $w \in \{a, b\}^*$ with $\mathbf{P}^F_{\mathcal{A}}(w) > \frac{1}{2}$ if and only if $Disc_{\max}(\mathsf{M}_{\mathcal{A}}) > \frac{1}{4}$. While similar to the one of Theorem 3, the proof is more involved because the strategies must be taken into account. ◀

As a consequence, we obtain:

▶ **Corollary 14.** *The maximal finite horizon disclosure of an MDP cannot be computed.*

Using a reduction of the value 1 problem in PA, we also have:

▶ **Theorem 15** (Undecidability of maximal finite horizon limit-sure disclosure)**.** *The maximal finite horizon limit-sure disclosure problem is undecidable for MDPs.*

Fortunately the maximal finite-horizon almost-sure disclosure problem is decidable. The proof relies on results for partially observable MDPs (POMDPs): a POMDP is an MDP where the strategies resolving the non determinism only depend on the observation sequence and do not take the current state into account.

▶ **Theorem 16** (Decidability of maximal finite-horizon almost-sure disclosure)**.** *The maximal finite-horizon almost-sure disclosure problem in MDPs is* EXPTIME*-complete. Moreover, if the system is almost-surely disclosing, one can build a belief-based strategy with disclosure* 1*.*

Sketch of Proof. We reduce the almost-sure disclosure problem for maximisation in MDPs to almost-sure reachability in a POMDP. The POMDP we build is exponential in the size of the original MDP and the algorithm to solve almost-sure reachability is exponential in the size of the POMDP [9]. This gives an EXPTIME algorithm as those two exponentials do not stack. The hardness is obtained by a reduction from the safety problem in games with imperfect information that was shown to be EXPTIME-complete in [6]. ◀

4 Minimisation with finite horizon

Recall from the example illustrated in Figure 1a of introduction, that randomised strategies are necessary for minimisation. To address this issue we introduce *families of almost deterministic strategies* based on *ε-decision rules*, that will be used in the decision procedures.

▶ **Definition 17.** Let δ be the deterministic decision rule for state s selecting action $a \in A(s)$. Then $\delta_\varepsilon \in \mathsf{Dist}(A(s))$ is a (randomised) ε-decision rule, said to *favour* a, and defined by:
1. If $|A(s)| > 1$ then $\delta_\varepsilon(a) = 1 - \varepsilon$ and for all $b \in A(s) \setminus \{a\}$, $\delta_\varepsilon(b) = \frac{\varepsilon}{|A(s)|-1}$;
2. Else $\delta_\varepsilon(a) = 1$.

▶ **Definition 18.** Let σ be an observation-based deterministic strategy. Then $\{\sigma_\varepsilon\}_{\varepsilon>0}$ is a family of *observation-based almost deterministic strategies* defined for any state s and $w \in \Sigma^n$, an observation of length $n \in \mathbb{N}$, by: $\sigma_\varepsilon(w, s) = \sigma(w, s)_{2^{-n}\varepsilon}$.

Using Proposition 11, we assume that the observation function O is non-erasing. The complexity of the transformation does not affect the results since the complexities are all polynomial in the number of actions. To compute the minimal disclosure value, we build from an MDP M, another MDP $\mathsf{M}_{\min}$ which is a "correct abstraction" (as stated by Proposition 19) for reducing minimal disclosure problems to minimal reachability problems, by enlarging states with the maximal belief that can occur independently of the action that has been selected.

Given a set of potential current states B and a new observation o, we define the maximal set of potential next states $\mathsf{NextMax}(B, o)$ over decision rules applied to B by:

$$\mathsf{NextMax}(B, o) = \{s' \in \mathsf{O}^{-1}(o) \mid \exists s \in B\ \exists a \in A(s)\ p(s'|s, a) > 0\}$$

Observe that given a family of almost deterministic strategies $\{\sigma_\varepsilon\}$ and a path ρas of M with $\mathsf{O}(s) = o$, one has $B^{\sigma_\varepsilon}_{\rho as} = \mathsf{NextMax}(B^{\sigma_\varepsilon}_{\rho}, o)$. Then $\mathsf{M}_{\min}$ is formally defined as follows:

- $S_{\min}$, the set of states, is defined by: $S_{\min} = \{(s, B) \mid s \in B \subseteq \mathsf{O}^{-1}(\mathsf{O}(s))\}$;
- Let $(s, B) \in S_{\min}$. Then $A(s, B) = A(s)$;
- Let $(s, B), (s', B') \in S_{\min}$. If $B' = \mathsf{NextMax}(B, \mathsf{O}(s'))$ then $p((s', B')|(s, B), a) = p(s'|s, a)$ else $p((s', B')|(s, B), a) = 0$.

Given μ_0 an initial distribution over S, the associated initial distribution $\mu_{\min}$ over $S_{\min}$ is defined by $\mu_{\min}(s, Supp(\mu_0) \cap \mathsf{O}^{-1}(\mathsf{O}(s))) = \mu_0(s)$ and $\mu_{\min}(s, B) = 0$ for all other B. We define the subset $\mathsf{Avoid}(Sec) \subseteq S_{\min}$ by $\mathsf{Avoid}(Sec) = \{(s, B)|B \subseteq Sec\}$.

▶ **Proposition 19.** *The minimal disclosure value for Sec in* $\mathsf{M}(\mu_0)$ *is equal to the minimal probability to reach* $\mathsf{Avoid}(Sec)$ *in* $\mathsf{M}_{\min}(\mu_{\min})$*. Furthermore it is asymptotically reached by a family of belief-based almost deterministic strategies.*

Since minimal reachability probability in MDPs can be computed in polynomial time we immediately obtain the first part of the next theorem. We establish the second part (PSPACE-hardness) in the proof of Theorem 23.

▶ **Theorem 20.** *The minimal disclosure value of* $\mathsf{M}(\mu_0)$ *can be computed in* EXPTIME*. The associated decision problem is* PSPACE*-hard.*

We now turn to the existence of a strategy that achieves the minimal value and establish that it can be analysed without additional complexity. The main ingredient of the proof is an equation system over states of the MDP whose unique solution is the minimal reachability probability vector.

Notations. Given μ a distribution over states and $\vec{\delta}$ a vector of decision rules over states in the support of μ, we define $\mathsf{NextDist}(\mu, \vec{\delta})$ the next distribution over S when applying $\vec{\delta}$ by:

$$\mathsf{NextDist}(\mu, \vec{\delta})(s') = \sum_{s \in Supp(\mu)} \mu(s) p(s'|s, \vec{\delta}[s]) \quad \text{for any } s' \in S.$$

For a distribution μ over S and $o \in \Sigma$, we write $\mu(o)$ for $\mu(\mathsf{O}^{-1}(o))$. If $Supp(\mu) \cap \mathsf{O}^{-1}(o) \neq \emptyset$, the relative distribution μ_o over $\mathsf{O}^{-1}(o)$ is defined by: $\mu_o(s) = \frac{\mu(s)}{\mu(o)}$ for $s \in \mathsf{O}^{-1}(o)$ and $\mu_o(s) = 0$ otherwise.

We have $Disc_{\min}(\mathsf{M}(\mu)) = \sum_{o \in \Sigma} \mu(o) Disc_{\min}(\mathsf{M}(\mu_o))$. For use in the next proof, we define $disc^*(\mathsf{M}(s, B))$ as the minimal disclosure value when starting in M in state s with belief B. Given some belief B and some decision rule vector $\vec{\delta}$ over B we introduce the possible successors of B when applying $\vec{\delta}$: $\mathsf{Next}(B, \vec{\delta}) = \{s' \mid \exists s \in B\ \exists a \in Supp(\vec{\delta}[s])\ p(s'|s, a) > 0\}$ and $\mathsf{Next}(B, \vec{\delta}, o) = \mathsf{Next}(B, \vec{\delta}) \cap \mathsf{O}^{-1}(o)$.

▶ **Theorem 21.** *The existence of a strategy that achieves the minimal disclosure value can be decided in* EXPTIME*. In the positive case, this strategy can be computed in* EXPTIME*.*

Proof. The algorithm simultaneously solves the existence and the synthesis problem.
Using proposition 19, the algorithm computes for all $(s, B) \in S_{\min}$, $disc^*(\mathsf{M}(s, B))$.
Then it maintains a set Win of beliefs initially set to all beliefs from which it iteratively eliminates items and stops when no more elimination is possible.
Given $B \in Win$, it looks for a decision rule vector $\vec{\delta}$ over B such that:

- for all $o \in \mathsf{O}(\mathsf{Next}(B, \vec{\delta}))$, $\mathsf{Next}(B, \vec{\delta}, o) \in Win$;
- for all $s \in B$, $disc^*(\mathsf{M}(s, B)) = \sum_{o \in \Sigma} \sum_{s' \in \mathsf{O}^{-1}(o)} p(s'|s, \vec{\delta}[s]) disc^*(\mathsf{M}(s', \mathsf{Next}(B, \vec{\delta}, o)))$.

If such a $\vec{\delta}$ does not exist then B is eliminated from Win. Each iteration can be performed in polynomial time w.r.t. $|S_{\min}|$ and the number of iterations is at most $|S_{\min}|$. Observe that when a belief is eliminated, it should not be "reached" by a strategy that obtains the minimal disclosure value. So the elimination is sound.

When the elimination stops, the algorithm answers positively iff for all $o \in \mathsf{O}(Supp(\mu_0))$, $Supp(\mu_0) \cap \mathsf{O}^{-1}(o) \in Win$. Thus by the soundness of the elimination step, if the answer is negative there is no optimal strategy for minimal disclosure value.

If the answer is positive, let us consider the belief-based strategy σ defined by applying the decision rules obtained during the last iteration of the algorithm. On the one hand, under σ when visiting a state s with belief B such that $disc^*(\mathsf{M}(s, B)) = 0$, one never leaves such kind of pairs of states and beliefs. So the secret is never disclosed, showing that the disclosure value obtained by σ for such (s, B) is null. Under σ the disclosure value of all the other pairs of state and belief fulfill the equations of the elimination step. It is known that the single solution of this system is the vector of minimal reachability probabilities of Avoid in $\mathsf{M}_{\min}(\mu_{\min})$ (see [1] for instance) which yields the result. ◀

5 Fixed horizon problems

5.1 Maximal disclosure

In order to compute the value of the maximal disclosure within a fixed horizon, one could build the POMDP described in the proof of Theorem 16 then use pre-existing results on POMDPs. This would result in an EXPTIME algorithm, whereas we obtain in the result below an algorithm with a better complexity in PSPACE.

▶ **Theorem 22** (Computation of the maximal disclosure value within fixed-horizon). *The fixed-horizon maximal value (when the horizon n is described in unary representation) is computable in* PSPACE *and the fixed-horizon maximal disclosure problem is* PSPACE*-complete.*

Sketch of proof - value and membership. We first order the observation alphabet Σ. Then a non deterministic decision procedure operating in PSPACE orderly reads every observation sequence of length n while maintaining the sets of states that were possible after every prefix of this observation, the actions that were chosen nondeterministically in those states and values used in the computation of the disclosure. The information kept is of polynomial size and when every observation has been read, one of the values computed will be exactly the disclosure of the system at time n. We then remove the non determinism using Savitch's Theorem. In order to get the value we observe that we can compute the polynomially sized denominator of this value and then we proceed by iterations of the decision algorithm. ◀

As can be seen in the proof, the optimal strategy could be computed when solving the value problem. However the size of this strategy may be exponential due to the beliefs and thus this strategy is computable in EXPTIME.

For the hardness result, we reduce the truth of a Quantified Boolean Formula (QBF). Recall that QBFs are extension of propositional formulas where boolean variables can be quantified. Syntactically, the formulas are described by the following grammar:

$$\phi ::= \psi \mid \exists x.\phi \mid \forall x.\phi$$
$$\psi ::= x \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid \texttt{true}$$

A QBF is *closed* if every boolean variable is bound by a quantifier. Deciding if a closed QBF is equivalent to `true` is PSPACE-hard [19].

Sketch of proof - hardness. Given ϕ a closed QBF (w.l.o.g. in 3CNF with n variables and m clauses), we build an MDP M such that ϕ is true iff the disclosure of M is greater or equal to $\frac{1}{2^{2n}}$ in $2(n+m)+3$ steps. In fact, $\frac{1}{2^{2n}}$ is exactly the measure of paths reaching the secret in $2(n+m)+3$ steps, thus every path reaching the secret must be disclosing. Such a path discloses the secret iff a boolean variable of ϕ and its negation (x and $\neg x$ for example) do not occur in its observation.

In M, during the first $2n$ steps, an assignment will be 'given' to each boolean variable: (i) for each existentially quantified boolean variable x, the strategy chooses whether x or $\neg x$ occurs in the observation and (ii) for each universally quantified boolean variable y, by a random choice with probability $\frac{1}{2}$. During the last $2m$ steps, the strategy must trigger a boolean variable in every clause of ϕ so that if a clause is not satisfied by the current assignment, then a boolean variable will be observed as both true and false during the path. Thus the observation would not disclose the secret. ◀

The existence of an optimal strategy here implies that the limit-sure and the almost-sure problem are equivalent. Moreover, the secret being revealed with probability 1 in a given number of steps implies that every path reaches the secret in this number of steps. Therefore the almost-sure problem can be seen as a reachability problem in an MDP which can be solved in polynomial time.

The proof of hardness can be adapted for ε-disclosure, but the algorithm for membership can not be directly applied. The ε-disclosure could however be computed by minimising an exponential system of equations, resulting in an exponential time algorithm.

5.2 Minimal disclosure

The proofs of the two first assertions of the next theorem are similar to the proof of Theorem 22. However in order to get the same complexity for the last assertion, we establish that when a randomised decision rule must be selected in the optimal strategy, it can always be uniformly distributed over its support.

▶ **Theorem 23** (Minimal disclosure within fixed horizon)**.** *The fixed horizon minimal value is computable in* PSPACE*. The fixed horizon minimal disclosure problem is* PSPACE*-complete. In addition, the strategy decision problem is also decidable in* PSPACE*.*

The above proof implies PSPACE-completeness for the limit-sure and almost-sure problem for minimisation. The remark on ε-disclosure of the previous subsection holds again here.

6 Conclusion

We revisit the problems of disclosure for MDPs by (1) taking into account general actions contrary to previous work and (2) considering both maximisation and minimisation problems. We almost fully characterise the decidability and complexity of those problems establishing an asymmetry between minimisation and maximisation problems: the former ones being easier although they require families of randomised strategies for reaching the optimal value.

There remains a complexity gap (PSPACE versus EXPTIME) for the finite-horizon minimisation problem that we want to fill. From a qualitative point of view, observe that disclosure is a hyperproperty as its truth value is defined relatively to a set of paths. Thus we plan to

address such kinds of properties in a restricted setting in order to get other decidability results. Another direction would be to strengthen the requirement for approximate ε-disclosure to regain decidability within finite horizon.

References

1 C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

2 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001. doi:10.1007/3-540-44647-8_1.

3 Béatrice Bérard, Krishnendu Chatterjee, and Nathalie Sznajder. Probabilistic opacity for markov decision processes. *Inf. Process. Lett.*, 115(1):52–59, 2015. doi:10.1016/j.ipl.2014.09.001.

4 Béatrice Bérard, Olga Kouchnarenko, John Mullins, and Mathieu Sassolas. Preserving opacity on interval markov chains under simulation. In Christos G. Cassandras, Alessandro Giua, and Zhiwu Li, editors, *13th International Workshop on Discrete Event Systems, WODES 2016, Xi'an, China, May 30 - June 1, 2016*, pages 319–324. IEEE, 2016. doi:10.1109/WODES.2016.7497866.

5 Béatrice Bérard, John Mullins, and Mathieu Sassolas. Quantifying opacity. *Mathematical Structures in Computer Science*, 25(2):361–403, 2015. doi:10.1017/S0960129513000637.

6 Dietmar Berwanger and Laurent Doyen. On the power of imperfect information. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India*, volume 2 of *LIPIcs*, pages 73–82. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1742.

7 Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *Int. J. Inf. Sec.*, 7(6):421–435, 2008. doi:10.1007/s10207-008-0058-x.

8 Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Thomas A. Henzinger. Randomness for free. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2010. doi:10.1007/978-3-642-15155-2_23.

9 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Qualitative analysis of partially-observable markov decision processes. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2010. doi:10.1007/978-3-642-15155-2_24.

10 Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval markov chains. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008. doi:10.1007/978-3-540-78499-9_22.

11 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and*

Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010. doi:10.1007/978-3-642-14162-1_44.

12 Jean Goubault-Larrecq and Roberto Segala. Random measurable selections. In Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten, editors, *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, volume 8464 of *Lecture Notes in Computer Science*, pages 343–362. Springer, 2014. doi:10.1007/978-3-319-06880-0_18.

13 D. J. D. Hughes and V. Shmatikov. Information Hiding, Anonymity and Privacy: a Modular Approach. *Journal of Computer Security*, 12(1):3–36, 2004.

14 Laurent Mazaré. Decidability of opacity with non-atomic keys. In Theodosis Dimitrakos and Fabio Martinelli, editors, *Formal Aspects in Security and Trust: Second IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), an event of the 18th IFIP World Computer Congress, August 22-27, 2004, Toulouse, France*, volume 173 of *IFIP*, pages 71–84. Springer, 2004. doi:10.1007/0-387-24098-5_6.

15 Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, 1987. doi:10.1287/moor.12.3.441.

16 A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

17 Anooshiravan Saboori and Christoforos N. Hadjicostis. Current-state opacity formulations in probabilistic finite automata. *IEEE Trans. Automat. Contr.*, 59(1):120–133, 2014. doi:10.1109/TAC.2013.2279914.

18 Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking markov chains in the presence of uncertainties. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, volume 3920 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006. doi:10.1007/11691372_26.

19 M. Sipser. *Introduction to the theory of computation*. Thomson Course Technology, 2006.

Reasons for Hardness in QBF Proof Systems*

Olaf Beyersdorff[1], Luke Hinde[2], and Ján Pich[3]

1 **School of Computing, University of Leeds, UK**
2 **School of Computing, University of Leeds, UK**
3 **Kurt Gödel Research Center, University of Vienna, Austria**

Abstract

We aim to understand inherent reasons for lower bounds for QBF proof systems, and revisit and compare two previous approaches in this direction.

The first of these relates size lower bounds for strong QBF Frege systems to circuit lower bounds via *strategy extraction* (Beyersdorff & Pich, LICS'16). Here we show a refined version of strategy extraction and thereby for any QBF proof system obtain a trichotomy for hardness: (1) via circuit lower bounds, (2) via propositional Resolution lower bounds, or (3) 'genuine' QBF lower bounds.

The second approach tries to explain QBF lower bounds through *quantifier alternations* in a system called relaxing QU-Res (Chen, ICALP'16). We prove a strong lower bound for relaxing QU-Res, which also exhibits significant shortcomings of that model. Prompted by this we propose an alternative, improved version, allowing more flexible oracle queries in proofs. We show that lower bounds in our new model correspond to the trichotomy obtained via strategy extraction.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases proof complexity, quantified Boolean formulas, resolution, lower bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.14

1 Introduction

Proof complexity studies the question of how difficult it is to prove theorems in different formal proof systems. The main question is thus: given a theorem ϕ and proof system P, what is the size of the shortest proof of ϕ in P? This research has strong and productive connections to several other areas, most notably to computational complexity, with the aim of separating complexity classes through Cook's programme [14, 11], and to first-order logic (theories of bounded arithmetic [26, 13]). In recent years, progress in practical SAT- and QBF-solving has been a major motivation for proof complexity, as runs of SAT-solvers correspond to proofs of the (un)satisfiability of CNFs. Analysis of the corresponding proof system provides a framework for understanding the power and limitations of the solver [11].

The majority of work in proof complexity has been focused on *propositional proof complexity*, on proof systems for classical propositional logic. In particular, Resolution [32] has received much attention as it models the approach taken by many modern SAT-solvers.

QBF proof complexity is a comparatively young field, studying proof systems for quantified Boolean formulas. Determining the truth of a QBF is PSPACE-complete, and so has wider ranging applications than SAT-solving, extending to fields such as formal verification and planning [3, 31, 15]. Similarly to the propositional case, several Resolution-based QBF

* Supported by grant no. 48138 from the John Templeton Foundation and by the Austrian Science Fund (FWF) under project number 28699.

proof systems have been suggested and analysed [24, 36, 1, 17, 5, 22, 7, 6, 35, 29] to model the approaches taken by QBF solvers. Of particular importance are Q-Resolution [24] and universal Q-Resolution (QU-Res) [17], which as analogues of propositional Resolution form the base systems for conflict-driven clause learning (CDCL) QBF solving [18].

Stronger systems in the form of QBF cutting planes [8] or QBF Frege systems [4] were developed recently by adding to the propositional system a single ∀-reduction rule to handle universal quantifiers. As in the propositional framework, by restricting the lines in Frege to a circuit class $\mathcal{C}$, such as AC^0, NC^1 or $P/poly$, we obtain a hierarchy of (QBF) $\mathcal{C}$-Frege systems, corresponding to the hierarchy of circuit classes.

A conceptually simple but powerful technique for constructing QBF proof size lower bounds from Boolean circuit lower bounds was developed in [6, 4]. This *strategy extraction technique* employs the complexity of Herbrand functions witnessing the universal quantifiers. In [4] the technique was used to show strong lower bounds for QBF Frege systems, including exponential lower bounds for QBF $AC^0[p]$-Frege (in stark contrast to the propositional situation, where lower bounds for $AC^0[p]$-Frege are wide open).

Recent work has tightened the connection to circuit complexity further. In [9] it was shown that for natural circuit classes $\mathcal{C}$, a lower bound for proof size in QBF $\mathcal{C}$-Frege corresponds to either a lower bound for propositional $\mathcal{C}$-Frege, or a lower bound for the circuit class $\mathcal{C}$. This characterisation points to a distinction between QBF lower bounds derived from those on propositional proof systems, and 'genuine' QBF lower bounds.

More widely, *understanding the reasons of hardness* for QBF proof systems and solving constitutes a major challenge, which currently is only insufficiently mastered. Most QBF proof systems use a propositional system such as Resolution or Frege as their core, implying that on existentially quantified formulas the QBF system and the classical core system coincide. This leads to the rather disturbing fact that lower bounds for e.g. Resolution trivially lift to any of the studied QBF Resolution systems.

Motivated by this observation, Chen [12] introduced the new notions of *relaxing QU-Res* and a *proof system ensemble*, with the aim of distinguishing 'genuine' QU-Res lower bounds, arising from the alternation of quantifiers, from those lifted from propositional Resolution. Quantifier alternation has also been empirically observed as a source of hardness [28, 27], making this a very interesting direction for theoretical study.

Our Contributions. The main aim of this paper is to gain a refined understanding of the reasons for QBF hardness, both following the strategy extraction paradigm [9] and the paradigm via quantifier alternation [12]. We revisit both models and relate them in their explanatory power.

A. Refinement of formalised strategy extraction. We describe a decomposition of QBF solving into SAT solving and a search for small circuits witnessing a given QBF. This relies on an improvement of the strategy extraction theorem from [9] which says that, given polynomial-size QBF $\mathcal{C}$-Frege proofs of QBFs ψ_n, one can construct small $\mathcal{C}$ circuits witnessing the existential quantifiers in ψ_n in such a way that the resulting 'witnessed' propositional formulas have polynomial-size proofs in $\mathcal{C}$-Frege. Here, we show that in fact the witnessed formulas have polynomial-size proofs even in tree-like Resolution (Theorem 1).

Applying a similar decomposition, we observe that polynomial-size lower bounds on a sequence of QBFs in any QBF proof system can be categorized as either (1) a circuit lower bound, (2) a Resolution lower bound, or (3) a genuine QBF lower bound (Theorem 2).

B. Lower bounds for relaxing QU-Res. We revisit relaxing QU-Res, introduced in [12] with the aim of distinguishing propositional bounds from QBF bounds arising from quantifier alternation. The exponential lower bound for relaxing QU-Res given in [12] applies only to quantified Boolean circuits with no small CNF representations (Appendix **??**). As this is a somewhat atypical feature in proof complexity, we improve this by presenting QBFs with CNF matrices that require exponential-size relaxing QU-Res proofs (Theorem 9). Our formulas use a new construction that combines two false QBFs Φ and Ψ into a product formula $\Phi \otimes \Psi$ such that any short QU-Res proof must refute Ψ before Φ.

These product formulas have another compelling feature: their hardness for relaxing QU-Res (and QU-Res) rests on the hardness of the pigeonhole principle for propositional Resolution. Our lower bound therefore suggests that relaxing QU-Res does not capture 'genuine' QBF hardness due to quantifier alternation.

C. New systems for 'genuine' QBF hardness. Noting this situation, we propose new QBF proof systems, Σ_k^p-QU-Res (Def. 15). The systems bear similarities to relaxing QU-Res, particularly in the use of relaxations of quantifiers and a proof checking algorithm with access to a Σ_k^p-oracle. The major difference is that oracle queries in our algorithm may appear at any point in the proof.

It is interesting to relate lower bounds in Σ_1^p-QU-Res to our trichotomy shown in A. In this direction, we prove that Σ_1^p-QU-Res admits strategy extraction by depth-3 Boolean circuits (Lemma 18). Hence QU-Res lower bounds stemming from circuit lower bounds (case (1) in the trichotomy in A) translate to lower bounds in Σ_1^p-QU-Res. Further, if a QBF is hard for QU-Res due to a Resolution lower bound (case (2) in A), it has short proofs in Σ_1^p-QU-Res. We also demonstrate that a variant of the prominent formulas of Kleine Büning et al. [24] simultaneously has genuine QBF lower bounds as per case (3) in A (Theorem 4) and is hard for Σ_k^p-QU-Res proofs for any constant k (Theorem 22).

Organisation. In Sec. 2 we detail necessary background. Section 3 refines formalised strategy extraction and the characterisation of QBF lower bounds from [9]. In Sec. 4 we show the lower bound for relaxing QU-Res. Section 5 contains the definition of Σ_k^p-QU-Res and a comparison of lower bounds in these systems with the characterisation in Sec. 3. In Sec. 6, we analyse the hardness of several QBF families in these proof systems.

2 Preliminaries

Quantified Boolean Formulas. A (prenex normal form) *quantified Boolean formula* (QBF) $\Phi = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n . \phi(x_1, \dots, x_n)$ consists of a propositional formula ϕ, usually expressed as a CNF, and a quantifier prefix $\mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n$, where each $\mathcal{Q}_i \in \{\exists, \forall\}$ ranges over $\{0, 1\}$.

The semantics of such a QBF can be considered as a game between players $\exists$ and $\forall$. On the ith turn, the player corresponding to $\mathcal{Q}_i$ assigns a 0/1 value to x_i. After all the variables have been assigned, the $\exists$ player (resp. $\forall$ player) wins the game if ϕ evaluates to 1 (resp. 0).

Given a variable x_i, a *strategy* for x_i is a function $\sigma_i : \{x_1, \dots, x_{i-1}\} \to \{0, 1\}$. A *winning strategy* for the $\exists$ (resp. $\forall$) player, consists of a strategy for each existential (resp. universal) variable which wins all possible games on Φ. A QBF is false (resp. true) if and only if there is a winning strategy for the $\forall$ player (resp. $\exists$ player).

The quantifier complexity of a QBF is described by inductively defined classes Σ_i^b and Π_i^b, counting the number of quantifier alternations. By Σ_i^p (resp. Π_i^p) we denote the i^{th} level of the polynomial hierarchy, for which deciding truth of Σ_i^b (resp. Π_i^b) formulas is complete.

Proof Complexity. A *proof system* for a language $\mathcal{L}$ is a polynomial-time computable surjective function $f : \{0,1\}^* \to \mathcal{L}$ [14]. If $f(\pi) = \phi$, we say π is an f-proof of ϕ. Given proof systems P and Q for $\mathcal{L}$, P *p-simulates* Q if there is a polynomial-time function t with $P(t(\pi)) = Q(\pi)$ for any π. Two proof systems are *p-equivalent* if they p-simulate each other.

We use proof systems for propositional tautologies and fully quantified true QBFs (and for unsatisfiable formulas and false QBFs; we use the words *proof* and *refutation* interchangeably).

Resolution [32] is one of the best studied propositional proof systems [34]. Given two clauses $C \vee x$ and $D \vee \neg x$, Resolution can derive the clause $C \vee D$. A Resolution proof that a CNF ϕ is unsatisfiable is a derivation of the empty clause $\bot$ using the resolution rule.

QU-Resolution (QU-Res) [17] is a natural extension of Resolution to QBFs. Given a QBF $\Phi = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n.\phi$, where ϕ is a CNF, a QU-Res refutation of Φ is a derivation of $\bot$ from the clauses of ϕ. It uses the Resolution rule (with the extra condition that deriving tautological clauses is not allowed) and the $\forall$-*reduction rule*, which from a clause $C \vee l$ with literal l on universal variable x_i (i.e., $l = x_i$ or $l = \neg x_i$) can derive the clause C provided C contains no literals on $x_{i+1}, \dots, x_n$.

A proof in Resolution (and QU-Res, and other proof systems) can be represented as a directed acyclic graph (dag) with a root labelled by $\bot$, and input vertices labelled with clauses from the CNF. If we restrict the dag to be a tree, we define *tree-like Resolution*, which we denote by R^*. Tree-like Resolution is known to be weaker than Resolution [10].

Frege Systems. Frege systems are common 'textbook' proof systems comprised of a set of axiom schemes and inference rules [14]. Lines of a Frege proof are formulas in propositional variables and Boolean connectives $\wedge, \vee, \neg$. A Frege proof of ϕ is a sequence of formulas, ending with ϕ, in which each formula is either a substitution instance of an axiom, or is inferred from previous formulas by a valid inference rule. We also consider refutational Frege systems, in which we start with the formula $\neg\phi$ and derive a contradiction.

For a given circuit class $\mathcal{C}$, we define $\mathcal{C}$-Frege, as in [23], to be a Frege system which works with lines consisting of circuits in $\mathcal{C}$ and a finite set of derivation rules. If $\mathcal{C}$ consists of all Boolean circuits, then $\mathcal{C}$-Frege is p-equivalent to extended Frege (EF). If $\mathcal{C}$ is restricted to Boolean formulas, i.e. $\mathcal{C} = NC^1$, then NC^1-Frege is Frege as defined above.

An elegant method for extending $\mathcal{C}$-Frege systems to QBF was shown in [4]. The QBF proof system $\mathcal{C}$-Frege+$\forall$-red is a refutational proof system working with circuits from $\mathcal{C}$. The inference rules of $\mathcal{C}$-Frege+$\forall$-red are those of $\mathcal{C}$-Frege, along with the $\forall$-red rule $\frac{L_j(u)}{L_j(u/B)}$, where u is quantified innermost among the variables of the proof line L_j with respect to the quantifier prefix, and the circuit B does not contain any variables right of u. Restricting the circuit B in the $\forall$-red rule to the constants $0, 1$ results in a p-equivalent system [9].

3 Strategy extraction and reasons for hardness

A QBF proof system P has the *strategy extraction property* if for any P-proof π of a QBF ψ of the general form $\forall x_1 \exists y_1 \dots \forall x_n \exists y_n.\, \phi(x_1, \dots, x_n, y_1, \dots, y_n)$, where ϕ is a propositional formula, there are $|\pi|^{O(1)}$-size circuits C_i witnessing the existential quantifiers in ψ, i.e.

$$\bigwedge_{i=1}^{n} (y_i \leftrightarrow C_i(x_1, \dots, x_i, y_1, \dots, y_{i-1})) \to \phi(x_1, \dots, x_n, y_1, \dots, y_n). \tag{1}$$

The strategy extraction is *Q-formalised* if, in addition, the propositional formulas (1) have $|\pi|^{O(1)}$-size proofs in a propositional proof system Q.

For any QBF ψ, either there is a propositional formula as in (1) equivalent to ψ, or there are no (small) circuits C_i witnessing the existential variables, and so no QBF proof system with the strategy extraction property can prove ψ feasibly.

The task of *QBF solving based on proof systems admitting strategy extraction is thus reducible to the task of finding the witnessing circuits C_i, and then SAT solving of the witnessed formula.* Alternatively, we can speak about a reduction of QBF solving to Σ^q_2-formulas with existentially quantified witnessing circuits: $\exists C_1, \ldots, C_n \forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ \bigwedge_{i=1}^n (y_i \leftrightarrow C_i(x_1, \ldots, x_i, y_1, \ldots, y_{i-1})) \rightarrow \phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$.

We will show that all QBF proof systems P p-simulated by EF+$\forall$-red[1] have R^*-formalized strategy extraction. More precisely, we improve the formalised strategy extraction for EF+$\forall$-red from [9] by observing that the witnessing circuits can encode extension variables, which allows us to replace the EF proof of the witnessed formula with an R^* proof.

Consequently, instead of determining whether there is a short P-proof of ψ, one can solve the equivalent problem of whether there are small circuits C_i and a short R^*-proof of (1). As R^* is quasi-automatisable (i.e., R^* refutations for a given CNF can be constructed in quasi-polynomial time in the size of the smallest R^* proof [2]), the problem is essentially reduced to the search for the right witnessing circuits C_i.

▶ **Theorem 1.** *Let $\mathcal{C}$ be the circuit class NC^1 or $P/poly$.*[2] *Given a $\mathcal{C}$-Frege+$\forall$-red refutation π of a QBF $\exists x_1 \forall y_1 \ldots \exists x_n \forall y_n.\, \phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ where $\phi \in \Sigma^b_0$, we can construct in time $|\pi|^{O(1)}$ an R^* refutation of the witnessed formula $\bigwedge_{i=1}^n (y_i \leftrightarrow C_i(x_1, \ldots, x_i, y_1, \ldots, y_{i-1})) \wedge \phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ for some circuits $C_i \in \mathcal{C}$.*

Proof. By the formalised strategy extraction theorem for $\mathcal{C}$-Frege systems [9], there is a $\mathcal{C}$-Frege proof of the witnessed formula (1). This means there is an R^* refutation of $Ext \wedge \bigwedge_{i=1}^n (y_i \leftrightarrow C_i(x_1, \ldots, x_i, y_1, \ldots, y_{i-1})) \wedge \phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ where Ext is a set of extension axioms defining $\mathcal{C}$ formulas on the variables $x_1, \ldots, x_n, y_1, \ldots, y_n$. With the exception of those depending on y_n, these axioms can be encoded into circuits C_i with each extension variable represented by a possibly redundant gate of a circuit C_i. In order to remove the extension variables depending on y_n, we construct two independent R^* refutations, one with all occurrences of y_n in clauses of Ext substituted by 0 and the other with occurrences of y_n in Ext substituted by 1. This results in two R^* derivations, both at most as large as the original, one concluding with $\{y_n\}$ and the other with $\{\neg y_n\}$. Resolving on these two clauses we obtain the needed R^* derivation without extension variables depending on y_n. ◀

The reduction of QBF solving to SAT solving presented above is also of use for proving QBF proof complexity lower bounds. In [9] it was shown that any super-polynomial lower bound on EF+$\forall$-red is either a super-polynomial circuit lower bound or a super-polynomial lower bound on EF. Here we generalise this phenomenon to other QBF proof systems.

Let P be a refutational QBF proof system operating on clauses of matrices of (prenex normal form) QBFs which contains a resolution rule that allows resolution on both existential and universal variables. We say that a set of clauses C defines a formula $C_i(\vec{x}) = z$ for a circuit C_i with input variables $\vec{x}$ and output variable z if z appears in a literal of some clause in C and for any assignment of the input variables there is exactly one assignment of the remaining variables satisfying all clauses in C.

[1] This includes all commonly studied Resolution-based QBF systems.

[2] The result easily generalises to further 'natural' circuit classes $\mathcal{C}$ such as AC^0 or TC^0, but we will focus here on the two most interesting cases NC^1 and $P/poly$ leading to Frege and EF systems, respectively.

Whenever a QBF ψ as above is hard for a QBF proof system P it is for one of the following reasons:

1. the existential quantifiers in ψ cannot be witnessed by circuits C_i such that formulas $\bigwedge_i C_i(x_1, \ldots, x_i, y_1, \ldots, y_{i-1}) = y_i$ have $|\phi|^{O(1)}$-size P-derivations from $\neg\phi$.
2. the existential quantifiers in ψ are witnessable as in 1. but the witnessed formula $\bigwedge_{i=1}^{n} (y_i \leftrightarrow C_i(x_1, \ldots, x_i, y_1, \ldots, y_{i-1})) \wedge \neg\phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ is hard for Resolution.

This characterisation can be specified further.

▶ **Theorem 2.** *Let P be a refutational QBF proof system as above admitting strategy extraction by $\mathcal{C}$ circuits. If $\psi_n = \forall x_1 \exists y_1 \ldots \forall x_n \exists y_n . \phi_n(x_1, \ldots, x_n, y_1, \ldots, y_n)$ are QBFs with propositional CNF ϕ_n, which do not have polynomial-size proofs in P, then one of the following holds:*

1. ***Circuit lower bound.*** *The existential variables in ψ_n are not witnessable by $\mathcal{C}$ circuits.*
2. ***Resolution lower bound.*** *Condition 1. does not hold, but for all $\mathcal{C}$ circuits witnessing ψ_n, the witnessed formulas require super-polynomial size Resolution refutations.*
3. ***Genuine QBF hardness.*** *There are circuits $C_i \in \mathcal{C}$ witnessing ψ_n so that the witnessed formulas have polynomial-size Resolution refutations, but for all such circuits C_i it is hard to derive $\bigwedge_i C_i(x_1, \ldots, x_i, y_1, \ldots, y_{i-1}) = y_i$ from $\neg\phi_n$ in P.*

This means that any QBF lower bound on P is either a circuit lower bound, a propositional proof complexity lower bound, or a 'genuine' QBF proof complexity lower bound in the sense that P cannot derive efficiently some circuits witnessing the existential quantifiers in the original formula and whenever it can do that for some other witnessing circuits, the witnessed formula is hard for Resolution.

The last possibility does not happen in the case of strong systems like EF+$\forall$-red [9]. The situation is, however, more delicate with weaker systems, where we can encounter 'genuine' QBF lower bounds. We give an example.

▶ **Definition 3** (Kleine Büning et al. [24])**.** The QBFs KBKF_n are defined as $\exists y_0 y_1 y_1' \forall x_1 \ldots \exists y_k y_k' \forall x_k \ldots \forall x_n \exists y_{n+1} \ldots y_{n+n} . \bigwedge_{i=1}^{2n} C_i \wedge C_i'$, where

$$\begin{aligned} C_0 &= \{\neg y_0\} & C_0' &= \{y_0, \neg y_1, \neg y_1'\} \\ C_k &= \{y_k, \neg x_k, \neg y_{k+1} \neg y_{k+1}'\} & C_k' &= \{y_k', x_k, \neg y_{k+1}, \neg y_{k+1}'\} \\ C_n &= \{y_n, \neg x_n, \neg y_{n+1}, \ldots, \neg y_{n+n}\} & C_n' &= \{y_n', x_n, \neg y_{n+1}, \ldots, y_{n+n}\} \\ C_{n+t} &= \{x_t, y_{n+t}\} & C_{n+t}' &= \{\neg x_t, y_{n+t}\} \end{aligned}$$

These QBFs are known to require proofs of size $2^{\Omega(n)}$ in Q-Resolution [24, 6]. This bound can be extended to QU-Res using the formulas KBKF_n', obtained by adding new universal variables z_k, quantified at the same level as x_k, and adding the literal z_k or $\neg z_k$ to each clause containing x_k or $\neg x_k$, respectively [1].

▶ **Theorem 4.** *The formulas KBKF_n' are hard for QU-Res due to genuine QBF hardness (case 3 in Theorem 2).*

Proof Sketch. The strategy $x_k = z_k = y_k'$ is a short winning strategy for the $\forall$-variables, and has a linear-size refutation by first deriving each y_{n+t}, then y_n' and y_n, and each y_i' and y_i in turn, concluding by resolving y_0 and $\neg y_0$. ◀

4 Hardness due to quantifier alternation

The characterisation of QBF proof system lower bounds given above is a very natural one. We now seek to show that it corresponds with hardness due to alternation, another often suggested reason for hardness.

Most studied QBF proof systems build on a propositional proof system (e.g. Resolution), and coincide with this base system on Σ_1^b-formulas. Any propositional lower bound is therefore also a QBF lower bound. An alternative characterisation of QBF lower bounds based on the alternation of quantifiers in the quantifier prefix has been suggested, with the aim of distinguishing between such propositional lower bounds and 'genuine' QBF lower bounds arising from the alternation of quantifiers. Relaxing QU-Res has previously been put forward as a proof system to determine hardness due to quantifier alternation [12].

▶ **Definition 5** (Chen [12]). For a quantifier prefix $\Pi = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n$, if π is a permutation such that $\pi(i) < \pi(j)$ whenever $i < j$ and $\mathcal{Q}_i = \forall$ and $\mathcal{Q}_j = \exists$, then the prefix $\Pi' = \mathcal{Q}_{\pi(1)} x_{\pi(1)} \dots \mathcal{Q}_{\pi(n)} x_{\pi(n)}$ is a *relaxation*. Intuitively, a relaxation involves 'moving $\forall$-variables to the left'. If Π' is a Σ_k^b-prefix, we call Π' a Σ_k^b*-relaxation*.

Let $\Phi = \Pi.\phi$ be a QBF. For a clause A, let α be the assignment falsifying each literal in A. Construct $\Pi[\alpha]$ by removing all variables in α, and replacing any $\forall$-quantifers left of a variable in α by $\exists$. If there is some Π_k^b-relaxation $\Pi'[\alpha]$ of $\Pi[\alpha]$ such that $\Pi'[\alpha].\phi[\alpha]$ is false, then $A \in H(\Phi, \Pi_k^b)$.

A *Relaxing QU-Res* proof of a QBF Φ uses the same deduction rules as QU-Res, but can introduce any axiom from the set $H(\Phi, \Pi_k^b)$ for some constant k.

For any propositional CNF, or indeed any QBF with a prefix with bounded alternation, relaxing QU-Res has constant-size proofs, whereas QU-Res may require exponential-size proofs. However, lower bounds for both tree-like and dag-like relaxing QU-Res were also shown in [12]. The lower bound for dag-like relaxing QU-Res in [12] is rather unconventional as the proof system works with clauses, whereas the lower bound applies to circuits without polynomial-size CNF representations. We present formulas with polynomially many clauses that require exponential-size proofs in relaxing QU-Res.

Furthermore, the lower bounds we show on the size of QU-Res proofs of these formulas are clearly due to a lower bound on Resolution proofs, rather than alternation of quantifiers, or any other 'genuine' QBF reasons. It follows that this is the case for relaxing QU-Res as well, demonstrating that relaxing QU-Res is not an adequate formalism to distinguish propositional lower bounds from genuine QBF lower bounds.

To begin, we present a method of combining two false QBFs to produce another false QBF. This method might also be of independent interest for the creation of hard QBFs.

▶ **Definition 6.** Let $\Phi = \Lambda(\vec{x}) \cdot \bigwedge_{i=1}^n C_i(\vec{x})$ and $\Psi = \Pi(\vec{z}) \cdot \bigwedge_{j=1}^m D_j(\vec{z})$ be QBFs consisting of quantifier prefixes Λ and Π over the disjoint sets of variables $\vec{x}$ and $\vec{z}$ respectively, and of clauses C_i and D_j over the corresponding variables. Then, with $\vec{x}$ and each $\vec{z}_i$ distinct variables, define

$$\Phi \otimes \Psi := \Lambda(\vec{x})\Pi(\vec{z}_1)\dots\Pi(\vec{z}_n) \cdot \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} (C_i(\vec{x}) \vee D_j(\vec{z}_i)) .$$

The QBF $\Phi \otimes \Psi$ is false if and only if Φ and Ψ are false. Combining winning strategies for the universal variables of Φ and Ψ constructs a strategy which falsifies some $C_i(\vec{x})$ and, for each i, falsifies some $D_j(\vec{z}_i)$. This strategy therefore falsifies some $C_i(\vec{x}) \vee D_j(\vec{z}_i)$. The

following lemma establishes that the proof size for $\Phi \otimes \Psi$ is bounded by the size of proofs required by Φ and Ψ, and indeed is precisely determined by it in the case of QU-Res.

▶ **Lemma 7.** *Let $\Phi = \vec{\mathcal{Q}}.\bigwedge_{i=1}^{n} C_i$ and $\Psi = \vec{\mathcal{S}}.\bigwedge_{j=1}^{m} D_j$ be minimally unsatisfiable QBFs. Let $S_P(\Lambda)$ be the size of the smallest P-proof of a QBF Λ. Then $\max(S_P(\Phi), S_P(\Psi)) \leq S_P(\Phi \otimes \Psi) \leq S_P(\Phi) + n \cdot S_P(\Psi)$. Moreover, if P is QU-Res, then $S_P(\Phi \otimes \Psi) = S_P(\Phi) + n \cdot S_P(\Psi)$.*

Proof. All clauses of $\Phi \otimes \Psi$ are necessary for a refutation. By assigning variables from Φ or the copies of Ψ appropriately, the lines in the proof can be restricted to a refutation of Φ or Ψ, and so $max(S_P(\Phi), S_P(\Psi)) \leq S_P(\Phi \otimes \Psi)$. Since $\Phi \otimes \Psi$ can be refuted by first deriving each clause C_i from $\bigwedge_{j=1}^{m}(C_i(\vec{x}) \vee D_j(\vec{z}_i))$, which can be done in $S_P(\Psi)$, and then refuting $\bigwedge_{i=1}^{n} C_i(\vec{x})$ with size $S_P(\Phi)$, we can find a refutation of $\Phi \otimes \Psi$ of size $S_P(\Phi) + n \cdot S_P(\Psi)$.

In QU-Res, each resolution step or $\forall$-reduction step is performed on only one variable, and so will only remain in one of the proofs of Φ or $\Psi(\vec{z}_i)$, being replaced by a weakening or trivial step in all others. Any QU-Res proof of $\Phi \otimes \Psi$ must therefore have size at least $S_P(\Phi) + n \cdot S_P(\Psi)$. By the upper bound shown above, there is equality. ◀

We use this method to construct a family of false QBFs that require exponential-size proofs in QU-Res. These QBFs are the product of propositional formulas hard for Resolution and of QBFs easy for QU-Res. By Lemma 7, it is clear that the hardness of this product is derived only from the propositional lower bound. Yet, these product formulas are also hard for relaxing QU-Res. The QBF is obtained by taking the product of the pigeonhole principle, defined below, and the formulas by Kleine Büning et al. [24] (Definition 3).

▶ **Definition 8.** The *pigeonhole principle* PHP_n^m, for m pigeons and n holes, is the CNF $\bigwedge_{i=1}^{m}(x_{i,1} \vee \cdots \vee x_{i,n}) \wedge \bigwedge_{j=1}^{m}\bigwedge_{1 \leq i_1 < i_2 \leq n}(\neg x_{i_1,j} \vee \neg x_{i_2,j})$.

For $m > n$, this is unsatisfiable, and for $m = n+1$ it is known that $2^{\Omega(n)}$ clauses are required to refute it in Resolution, and indeed in any constant-depth Frege system [19, 30, 25].

▶ **Theorem 9.** *Any relaxing QU-Res proof of $\Phi_n := \mathrm{PHP}_n^{n+1} \otimes \mathrm{KBKF}_n$ has size $2^{\Omega(n)}$.*

When restricted to a propositional formula, QU-Res is equivalent to Resolution, and PHP_n^{n+1} requires proofs of size $2^{\Omega(n)}$ in Resolution [19], so PHP_n^{n+1} requires QU-Res proofs of size at least $2^{\Omega(n)}$. In QU-Res, the formulas KBKF_n have linear-size proofs [17]. Given the precise determination of the size of QU-Res proofs of Φ_n by Lemma 7, this QU-Res lower bound for Φ_n is unambiguously due to the lower bound for PHP_n^{n+1} in Resolution.

We first note that any relaxation of KBKF_n is true.

▶ **Lemma 10.** *Any relaxation of the quantifier prefix of KBKF_n to a Π_t^b prefix results in a true QBF, for any $t < n$.*

Proof Sketch. Any relaxation must quantify some x_i left of y_i, y_i'. The winning strategy for each x_i depends on y_i, y_i' and so there is a winning strategy for the existential player by playing y_i, y_i' according to x_i. ◀

Any clause in the variables of Φ_n can be written as $X \vee Z_1 \vee \cdots \vee Z_m$ where X is a clause in the variables of $\vec{x}$, and Z_i is a clause in the variables of $\vec{z}_i$. We use the terms Z-variables and X-variables to refer to any variables in $\vec{z}_1, \ldots, \vec{z}_m$ and $\vec{x}$ respectively. Similarly, given a clause C, we use X-clause and Z-clause to refer to the restriction of C to the X-variables and Z-variables, and denote these by C^X and C^Z.

To prove Theorem 9, we show that if the oracle deriving axioms 'proves' a large part of the pigeonhole principle when deriving an axiom, it can only do so under a large restriction on

the existential variables of the copies of KBKF_n. Thus a relaxing QU-Res proof of Φ_n must contain a large proof of the pigeonhole principle, or a large number of different restrictions on the copies of KBKF_n.

To this end, we first show that, for any clause A derived as an axiom by relaxing QU-Res, if A^X requires at least c clauses from PHP_n^{n+1} to prove, then it also contains at least c existentially quantified Z-variables (Lemma 11). We then establish an upper bound on the size of a proof of an X-clause derived from c axioms of PHP_n^{n+1} which depends only on c (Lemma 12). Using this, we conclude that any relaxing QU-Res axiom where the corresponding X-clause requires proofs of size 2^k must contain $\Omega(k)$ Z-variables (Corollary 13).

Lastly, we show that given any relaxing QU-Res proof, for each assignment to the Z-variables, we can find an axiom containing $\Omega(n)$ Z-variables which agrees with the given Z-assignment (Lemma 14). From this, we conclude that the proof must contain $2^{\Omega(n)}$ axioms.

▶ **Lemma 11.** *Suppose that the clause $A = A^X \vee A^Z$ is derived as an axiom of Φ_n by relaxing QU-Res. Let $Z_{i_1}, \dots, Z_{i_l}$ be such that all the existential variables in A^Z are in some Z_{i_j}. Then $C_{i_1} \wedge \dots \wedge C_{i_l} \models A^X$ for the corresponding pigeonhole principle axioms $C_{i_1}, \dots, C_{i_l}$.*

Proof Sketch. If not, there is some smallest assignment α satisfying $C_{i_1} \wedge \dots \wedge C_{i_l} \wedge \neg A$. For any Π_k^b-relaxation Φ', we extend this to a winning strategy for the existential player. On variables in Z_{i_j} for some $1 \leq j \leq l$, the strategy is arbitrary as α already satisfies C_{i_j}. On variables in Z_k where $k \neq i_j$ for any $1 \leq j \leq l$, no variables from Z_k are defined in α so we use a winning strategy for the relevant relaxation of KBKF_n in the variables of Z_k.

As no Π_k^b-relaxation of $\Phi[\alpha]$ is true, A is not an axiom in relaxing QU-Res. ◀

By Lemma 11, if relaxing QU-Res derives an axiom A, and any proof of A^X requires l axioms from PHP_n^{n+1}, then A contains existential variables from l different Z_i. In particular, A contains at least l distinct Z-variables.

Lemma 12 gives an upper bound for the size of Resolution proofs from a fixed number of axioms from PHP_n^{n+1}. From this and Lemma 11 follows the key observation for the proof of Theorem 9, that for any relaxing QU-Res axiom A, if A^X requires a large Resolution derivation from the pigeonhole principle then A contains a large number of Z-variables.

▶ **Lemma 12.** *Suppose C is a clause such that $C_1 \wedge \dots \wedge C_t \models C$ for some axioms $C_1, \dots, C_t$ from PHP_n^{n+1}. Then there is a Resolution proof of C from PHP_n^{n+1} of size at most 18^t.*

Proof Sketch. As all negative literals are in axioms of width two, the axioms contain positive and negative literals on at most $2t$ variables. Each axiom may also contain a set of pure literals, so there are at most $3^{2t}2^t$ derivable clauses. ◀

▶ **Corollary 13.** *Let A be an axiom derived from Φ_n by relaxing QU-Res. Let $S(A^X)$ be the size of the smallest Resolution derivation of A^X from PHP_n^{n+1}. Then A contains at least $\frac{1}{\log 18} \log S(A^X)$ existential Z-variables.*

▶ **Lemma 14.** *Given a relaxing QU-Res proof π and an assignment α to the existential Z-variables of Φ_n, $\pi|_\alpha^X$ contains a sound Resolution refutation of the X-axioms corresponding to axioms agreeing with α.*

Proof of Theorem 9. Suppose that π is a relaxing QU-Res proof of Φ_n with $|\pi| = f(n)$. Given an assignment α to the existential Z-variables, $\pi|_\alpha^X$ is a sound Resolution refutation of the X-axioms (Lemma 14), and has at most $f(n)$ axioms. Since any Resolution refutation of PHP_n^{n+1} requires proofs of size at least 2^{kn} for some constant k, some X-axiom B in $\pi|_\alpha^X$ requires a Resolution derivation of size at least $\frac{2^{kn}-f(n)}{f(n)} = \frac{2^{kn}}{f(n)} - 1$. By Corollary 13, there

is an axiom A in π such that $A^X = B$, and so A contains at least $c(kn - \log f(n)) =: g(n)$ existential Z-variables, which agree with α.

For every assignment α to the existential Z-variables, we can find such an axiom containing at least $g(n)$ existential Z-variables and agreeing with α. As each of these axioms can agree with at most a $2^{-g(n)}$ proportion of the possible assignments α, π must contain at least $2^{g(n)}$ axioms. As a proof cannot contain more axioms than its length, we conclude that $2^{g(n)} \leq f(n)$, i.e. $2^{ckn} \leq f(n)2^{c \log f(n)} = f(n)^{c+1}$ and so $f(n) = 2^{\Omega(n)}$. Thus any relaxing QU-Res proof of Φ_n has size $2^{\Omega(n)}$. ◀

We have shown that $\mathrm{PHP}_n^{n+1} \otimes \mathrm{KBKF}_n$ requires relaxing QU-Res proofs of size $2^{\Omega(n)}$, despite consisting of a hard propositional formula for Resolution combined with a QBF which is easy for QU-Res. This strengthens previous lower bounds for relaxing QU-Res, as well as demonstrating a QBF which is hard due to Resolution but also hard for relaxing QU-Res.

5 An alternative definition of hardness from alternation

We now define a family of proof systems which do characterise whether a QBF lower bound is due to quantifier alternation, or due to a propositional lower bound. As expected, $\mathrm{PHP}_n^{n+1} \otimes \mathrm{KBKF}_n$ have short proofs in these proof systems.

▶ **Definition 15.** A Σ_k^p-QU-Res proof of a QBF $\Phi = \Pi.\phi$ is a derivation of the empty clause by any of the rules of QU-Res, or the Σ_k^p-*derivation* rule $\frac{C_1 \quad \dots \quad C_l}{D}$ for any l, where there is some Σ_k^b-relaxation Π' of the quantifier prefix Π such that $\Pi'. \bigwedge_{i=1}^{l} C_i \models \Pi'.D \wedge \bigwedge_{i=1}^{l} C_i$.

In the context of these proof systems, we define a Σ_k^b-relaxation of a quantifier prefix as in Definition 5, i.e. any movement of $\forall$-variables to the left, but also allow replacing any $\forall$ quantifier by $\exists$. Allowing this replacement is not necessary, but, as shown in Lemma 19, it allows us to restrict our attention to Σ_{2k+1}^p-QU-Res, eliminating the need for a similarly defined Π_m^p-QU-Res.

It is straightforward to define Σ_k^p-P similarly for any line-based QBF proof system P, and several of the following results for QU-Res have analogues in these systems.

Any QU-Res proof is also a Σ_k^p-QU-Res proof, so Σ_k^p-QU-Res is complete. For soundness, note that QU-Res (with weakening) is both sound and inferentially complete [17]. Thus we can replace any Σ_k^p-derivation with a QU-Res derivation consistent with the Σ_k^b-relaxation. This QU-Res derivation will also be consistent with the original quantifier prefix, and so from any Σ_k^p-QU-Res refutation, we can construct such a QU-Res refutation. Since QU-Res is sound, Σ_k^p-QU-Res is therefore also sound.

We can now give our definition of hardness due to quantifier alternation.

▶ **Definition 16.** A family of QBFs is *hard due to quantifier alternation* if it requires superpolynomial-size Σ_1^p-QU-Res refutations.

A QBF family has *alternation hardness* Σ_k^p if it has polynomial-size proofs in Σ_k^p-QU-Res, but requires superpolynomial-size proofs in Σ_{k-1}^p-QU-Res.

The proof complexity of QBFs in Σ_1^p-QU-Res is of particular interest, as recent success in SAT solving has led to some QBF solvers embedding a SAT solver as a black box [33, 21]. The Σ_1^p-oracle access models this technique, and may provide insights into the power and limitations of such QBF solvers. As discussed below, proofs in Σ_1^p-QU-Res also characterise whether a QBF is hard due to a propositional lower bound, or whether the lower bound is, at least in part, derived from the use of universal quantifiers.

In Section 4, the formulas $\mathrm{PHP}_n^{n+1} \otimes \mathrm{KBKF}_n$ were shown to require QU-Res proofs of size $2^{\Omega(n)}$ due to a lower bound on Resolution. However, there are short Σ_1^p-QU-Res proofs of these formulas, even with only a single Σ_1^p-derivation, demonstrating that they are not hard for QU-Res due to quantifier alternation. This is in sharp contrast with the lower bound shown in Theorem 9, despite relaxing QU-Res also using Σ_k^p-oracles.

▶ **Theorem 17.** $\mathrm{PHP}_n^{n+1} \otimes \mathrm{KBKF}_n$ *have* Σ_1^p*-QU-Res proofs of length* $O(n^3)$.

Proof Sketch. The refutation first derives each of the $O(n^2)$ axioms of PHP_n^{n+1} in $O(n)$ lines in QU-Res, then derives the empty clause with a Σ_1^p-derivation. ◀

Any clause derived as an axiom of Φ using a Σ_k^p-oracle by relaxing QU-Res can also be derived from the clauses of Φ by a single Σ_k^p-derivation in Σ_k^p-QU-Res. It is easy to see from this that the instance of relaxing QU-Res using Σ_k^p-oracles is p-simulated by Σ_k^p-QU-Res. Theorems 9 and 17 show an exponential separation between even Σ_1^p-QU-Res and relaxing QU-Res for any k.

In order to compare the characterisation of lower bounds by quantifier alternation with the characterisation given in Section 3, we first show that Σ_1^p-QU-Res still has the same strategy extraction property as QU-Res. Analogous strategy extraction results apply for all $\mathcal{C}$-Frege+$\forall$-Red systems.

▶ **Lemma 18.** Σ_1^p*-QU-Res admits strategy extraction by depth-3 Boolean circuits.*

Proof Sketch. The Σ_1^p-derivations can be replaced by Resolution derivations. Strategy extraction in QU-Res is done with circuits polynomial in the number of $\forall$-reduction steps [4], and so polynomial in the size of the Σ_1^p-QU-Res proof. ◀

As a consequence of Lemma 18, QBFs hard for QU-Res by item 1 of Theorem 2 (strategy extraction) are therefore hard for Σ_1^p-QU-Res. Intuitively, we expect strategy extraction lower bounds to also be lower bounds due to alternation, as the strategy extraction technique inherently relies on universally quantified variables and the order of quantification.

Consider now QBFs hard for QU-Res by item 2 in Theorem 2. There are polynomial-size strategies for the universal variables, but for all of these, the witnessed formulas require superpolynomial-size proofs in Resolution. Using the normal form for proofs described in [9], we can construct short proofs of these QBFs in Σ_1^p-QU-Res, deriving the witnessed formula, then using a Σ_1^p-derivation to derive $\bot$. This demonstrates that QBFs hard by item 2 are not hard due to quantifier alternation.

For sufficiently strong proof systems, such as Frege+$\forall$-red, these are the only two reasons for hardness [9]. As Lemma 18 extends naturally to Σ_1^p-Frege+$\forall$-red, the characterisation of hardness for QBF Frege systems in [9] (circuit lower bounds vs propositional Frege lower bounds) therefore coincides with our characterisation via quantifier alternation.

6 Alternation Hardness of Specific Formulas

Finally, we determine the precise alternation hardness of specific families of QBFs from each of the categories defined in Theorem 2. Not all formulas from the same category necessarily have the same alternation hardness, but the bounds shown here reinforce the distinctions from Theorem 2.

The first step in establishing the alternation hardness of QBFs is to understand which levels we need to consider. The definition of relaxation allows replacing universal quantifiers with existential quantifiers, so we need only consider proof size in Σ_k^p-QU-Res for odd k.

▶ **Lemma 19.** *If a family of QBFs has proofs of size $s(n)$ in Π^p_m-QU-Res or Σ^p_{2k}-QU-Res, then it has proofs of size $n \cdot s(n)$ in Σ^p_{m-1}-QU-Res or Σ^p_{2k-1}-QU-Res respectively. Given a family of QBFs Φ_n, if the alternation hardness of Φ_n is $\mathcal{C}$, then $\mathcal{C} = \Sigma^b_{2k+1}$ for some k.*

Proof Sketch. A Σ^p_{2k}-derivation can be replaced by $\forall$-reduction on the rightmost variables, then a Σ^p_{2k-1}-derivation which quantifies the previously rightmost $\forall$-variables existentially. A Π^p_k-derivation can be replaced by a Σ^p_{k-1}-derivation, with the leftmost $\forall$-variables quantifed existentially, followed by a $\forall$-reduction if necessary. ◀

If the definition of relaxation were restricted to that given in the definition of relaxing QU-Res, by not allowing additional existential quantifiers, then the simulation of Π^p_m-QU-Res by Σ^p_{m-1}-QU-Res would not hold. With the exception of Σ^p_2-QU-Res, it would still be possible to reduce a Σ^p_{2k}-QU-Res proof to a Σ^p_{2k-1}-QU-Res proof by moving the rightmost universal variables leftwards to another block of universal quantifiers.

Lemmas 18 and 19 allow us to determine the precise alternation hardness of QParity_n, which were introduced in [6] as examples of formulas which are hard due to strategy extraction (item 1 in Theorem 2).

▶ **Definition 20** ([6])**.** The formulas QParity_n consist of the quantifier prefix $\exists x_1 \dots x_n \forall z \exists t_2 \dots t_n$ and clauses expressing that $t_2 \equiv x_1 \oplus x_2$, $t_k \equiv t_{k-1} \oplus x_k$ for each $3 \leq k \leq n$, and $z \equiv \neg t_n$.

The QBFs QParity_n are false, and the only winning strategy for the $\forall$-player is to play $z \equiv \bigoplus_{i=1}^n x_i$. As the parity function is hard to compute for depth-3 circuits [16, 20], any QU-Res proof has size $2^{\Omega(n)}$. The formulas QParity_n are therefore hard due to strategy extraction as defined in Theorem 2.

▶ **Corollary 21.** *The formulas QParity_n have Σ^b_3-alternation hardness.*

That the formulas QParity_n have Σ^b_3-alternation hardness shows that they are hard for QU-Res due to the alternation of quantifiers.

It is clear that all formulas which fall under item 2 of Theorem 2, of being hard only due to a lower bound on Resolution, such as PHP^{n+1}_n, have polynomial-size proofs in Σ_1-QU-Res, and so have Σ^p_1-alternation hardness.

The last family of QBFs we consider is KBKF'_n. By Theorem 4, the formulas KBKF'_n are hard for QU-Res due to a genuine QBF lower bound. As their hardness does not originate from a Resolution lower bound, we expect them to be hard due to alternation. In fact, we can go further and show that KBKF'_n are hard for Σ^p_k-QU-Res for all k.

▶ **Theorem 22.** *The formulas KBKF'_n require proofs of size $2^{\Omega(n)}$ in Σ^p_k-QU-Res for any constant k.*

▶ **Lemma 23.** *If a clause derived from KBKF'_n contains a literal on x_i, and the derivation does not contain a $\forall$-reduction step on x_i, then it contains y_j or y'_j for some $i \leq j \leq 2n$.*

Proof Sketch of Theorem 22. In the expansion to a QU-Res proof, no resolution steps are possible on universal pivots before these variables could be $\forall$-reduced.

For all assignments α to the universal variables, there is a $\forall$-reduction on each x_i containing literals agreeing with α on all universal variables left of x_i. By Lemma 23, these $\forall$-reductions can be chosen to contain a literal on y_i or y'_i. Thus at most k consecutive such $\forall$-reductions can be contained in one Σ^p_k-derivation, and so each of the 2^{n-k} assignments to $x_1, \dots, x_{n-k}$ must appear in full in a clause in the Σ^p_k-QU-Res proof. ◀

7 Conclusion

We have analysed both strategies and alternation as underlying reasons for the size of proofs in QBF proof systems. In the search for 'genuine' QBF lower bounds, these are the two characterisations which have received the most attention. We have shown that, for sufficiently strong proof systems (Frege and above), the two criteria are equivalent, and provided a natural proof system for which all lower bounds are such proper QBF lower bounds.

A natural question is whether for weaker Resolution-based systems, QBFs hard due to item 3 of Theorem 2 are always hard due to alternation. We have shown this only for the case of KBKF'_n in QU-Res. We also leave open the question of finding formulas which have alternation hardness precisely Σ^b_k for odd $k > 3$.

References

1 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2014. doi:10.1007/978-3-319-09284-3_12.

2 Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 274–282. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548486.

3 Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

4 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: From circuits to QBF proof systems. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 249–260. ACM, 2016. doi:10.1145/2840728.2840740.

5 Olaf Beyersdorff, Leroy Chew, and Mikolas Janota. On unification of QBF resolution-based calculi. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 81–93. Springer, 2014. doi:10.1007/978-3-662-44465-8_8.

6 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 76–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.76.

7 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Feasible interpolation for QBF resolution calculi. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2015. doi:10.1007/978-3-662-47672-7_15.

8 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Understanding cutting planes for qbfs. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65

of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.40.

9 Olaf Beyersdorff and Ján Pich. Understanding gentzen and frege systems for QBF. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 146–155. ACM, 2016. doi:10.1145/2933575.2933597.

10 Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM J. Comput.*, 30(5):1462–1484, 2000. doi:10.1137/S0097539799352474.

11 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012. doi:10.1016/j.apal.2011.09.009.

12 Hubie Chen. Proof complexity modulo the polynomial hierarchy: Understanding alternation as a source of hardness. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 94:1–94:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.94.

13 Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.

14 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979. doi:10.2307/2273702.

15 Uwe Egly, Martin Kronegger, Florian Lonsing, and Andreas Pfandler. Conformant planning as a case study of incremental QBF solving. In Gonzalo A. Aranda-Corral, Jacques Calmet, and Francisco J. Martín-Mateos, editors, *Artificial Intelligence and Symbolic Computation - 12th International Conference, AISC 2014, Seville, Spain, December 11-13, 2014. Proceedings*, volume 8884 of *Lecture Notes in Computer Science*, pages 120–131. Springer, 2014. doi:10.1007/978-3-319-13770-4_11.

16 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

17 Allen Van Gelder. Contributions to the theory of practical quantified boolean formula solving. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 647–663. Springer, 2012. doi:10.1007/978-3-642-33558-7_47.

18 Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified boolean formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 761–780. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-761.

19 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

20 Johan Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation, Advances in Computing Reasearch, Vol 5*, pages 143–170. JAI Press, 1989.

21 Mikolás Janota, William Klieber, Joao Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016. doi:10.1016/j.artint.2016.01.004.

22 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015. doi:10.1016/j.tcs.2015.01.048.

23 Emil Jerábek. Dual weak pigeonhole principle, boolean complexity, and derandomization. *Ann. Pure Appl. Logic*, 129(1-3):1–37, 2004. doi:10.1016/j.apal.2003.12.003.

24 Hans Kleine B"uning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995. doi:10.1006/inco.1995.1025.

25 Jan Krajícek, Pavel Pudlák, and Alan R. Woods. An exponenetioal lower bound to the size of bounded depth frege proofs of the pigeonhole principle. *Random Struct. Algorithms*, 7(1):15–40, 1995. doi:10.1002/rsa.3240070103.

26 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1995.

27 Florian Lonsing and Uwe Egly. Evaluating QBF solvers: Quantifier alternations matter. *CoRR*, abs/1701.06612, 2017.

28 Florian Lonsing, Uwe Egly, and Martina Seidl. Q-resolution with generalized axioms. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 435–452. Springer, 2016. doi:10.1007/978-3-319-40970-2_27.

29 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Long distance q-resolution with dependency schemes. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 500–518. Springer, 2016. doi:10.1007/978-3-319-40970-2_31.

30 Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3:97–140, 1993. doi:10.1007/BF01200117.

31 Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1045–1050, 2007.

32 John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

33 Horst Samulowitz and Fahiem Bacchus. Using SAT in QBF. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 578–592. Springer, 2005. doi:10.1007/11564751_43.

34 Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.

35 Friedrich Slivovsky and Stefan Szeider. Soundness of q-resolution with dependency schemes. *Theor. Comput. Sci.*, 612:83–101, 2016. doi:10.1016/j.tcs.2015.10.020.

36 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449. ACM / IEEE Computer Society, 2002. doi:10.1145/774572.774637.

An Improved Dictatorship Test with Perfect Completeness

Amey Bhangale[1], Subhash Khot[2], and Devanathan Thiruvenkatachari[3]

1 **Rutgers University, New Brunswick, USA**
arb182@cs.rutgers.edu
2 **New York University, New York, USA**
khot@cs.nyu.edu
3 **New York University, New York, USA**
t.devanathan@gmail.com

Abstract

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a dictator if it depends on exactly one variable i.e $f(x_1, x_2, \ldots, x_n) = x_i$ for some $i \in [n]$. In this work, we study a k-query dictatorship test. Dictatorship tests are central in proving many hardness results for constraint satisfaction problems.

The dictatorship test is said to have *perfect completeness* if it accepts any dictator function. The *soundness* of a test is the maximum probability with which it accepts any function far from a dictator. Our main result is a k-query dictatorship test with perfect completeness and soundness $\frac{2k+1}{2^k}$, where k is of the form $2^t - 1$ for any integer $t > 2$. This improves upon the result of [25] which gave a dictatorship test with soundness $\frac{2k+3}{2^k}$.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Property Testing, Distatorship Test, Fourier Analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.15

1 Introduction

Boolean functions are the most basic objects in the field of theoretical computer science. Studying different properties of Boolean functions has found applications in many areas including hardness of approximation, communication complexity, circuit complexity etc. In this paper, we are interested in studying Boolean functions from a property testing point of view.

In *property testing*, one has given access to a function $f : \{0,1\}^n \to \{0,1\}$ and the task is to decide if a given function has a particular property or whether it is *far* from it. One natural notion of farness is what fraction of f's output we need to change so that the modified function has the required property. A verifier can have an access to random bits. This task of property testing seems trivial if we do not have restrictions on how many queries one can make and also on the computation. One of the main questions in this area is can we still decide if f is very far from having the property by looking at a very few locations with high probability.

There are few different parameters which are of interests while designing such tests including the amount of randomness, the number of locations queried, the amount of computation the verifier is allowed to do etc. The test can either be *adaptive* or *non-adaptive*. In an adaptive test, the verifier is allowed to query a function at a few locations

and based on the answers that it gets, the verifier can decide the next locations to query whereas a non-adaptive verifier queries the function in one shot and once the answers are received makes a decision whether the function has the given property. In terms of how good the prediction is we want the test to satisfy the following two properties:

- **Completeness:** If a given function has the property then the test should accept with high probability
- **Soundness:** If the function is far from the property then the test should accept with very tiny probability.

A test is said to have *perfect completeness* if in the completeness case the test always accepts. A test with *imperfect completeness* (or almost perfect completeness) accepts a dictator function with probability arbitrarily close to 1. Let us define the soundness parameter of the test as how small we can make the acceptance probability in the soundness case.

A function is called a *dictator* if it depends on exactly one variable i.e $f(x_1, x_2, \ldots, x_n) = x_i$ for some $i \in [n]$. In this work, we are interested in a non-adaptive test with perfect completeness which decides whether a given function is a dictator or far from it. This was first studied in [4, 20] under the name of Dictatorship test and Long Code test. Apart from a natural property, dictatorship test has been used extensively in the construction of probabilistically checkable proofs (PCPs) and hardness of approximation.

An instance of a *Label Cover* is a bipartite graph $G((A, B), E)$ where each edge $e \in E$ is labeled by a projection constraint $\pi_e : [L] \to [R]$. The goal is to assign labels from $[L]$ and $[R]$ to vertices in A and B respectivels so that the number of edge constraints satisfied is maximized. Let $\texttt{GapLC}(1, \epsilon)$ is a promise gap problem where the task is to distinguish between the case when all the edges can be satisfied and at most ϵ fraction of edges are satisfied by any assignment. As a consequence of the PCP Theorem [1, 2] and the Parallel Repetition Theorem[22], $\texttt{GapLC}(1, \epsilon)$ is NP-hard for any constant $\epsilon > 0$. In [7], Håstad used various dictatorship tests along with the hardness of Label Cover to prove optimal inapproximability results for many constraint satisfaction problems. Since then dictatorship test has been central in proving hardness of approximation.

A dictatorship test with k queries and P as an accepting predicate is usually useful in showing hardness of approximating Max-P problem. Although this is true for many CSPs, there is no black-box reduction from such dictatorship test to getting inapproximability result. One of the main obstacles in converting dictatorship test to NP-hardness result is that the constraints in Label Cover are d-to-1 where the the parameter d depends on ϵ in $\texttt{GapLC}(1, \epsilon)$. To remedy this, Khot in [12] conjectured that a Label Cover where the constraints are 1-to-1, called *Unique Games*, is also hard to approximate within any constant. More specifically, Khot conjectured that $\texttt{GapUG}(1 - \epsilon, \epsilon)$, an analogous promise problem for Unique Games, is NP-hard for any constant $\epsilon > 0$. One of the significance of this conjecture is that many dictatorship tests can be composed easily with $\texttt{GapUG}(1-\epsilon, \epsilon)$ to get inapproximability results. However, since the Unique Games problem lacks perfect completeness it cannot be used to show hardness of approximating *satisfying* instances.

From the PCP point of view, in order to get k-bit PCP with perfect completeness, the first step is to analyze k-query dictatorship test with perfect completeness. For its application to construction PCPs there are two important things we need to study about the dictatorship test. First one is how to compose the dictatorship test with the known PCPs and second is how sound we can make the dictatorship test. In this work, we make a progress in understanding the answer to the later question. To make a remark on the first question, there is a dictatorship test with perfect completeness and soundness $\frac{2^{\tilde{O}(k^{1/3})}}{2^k}$ and also a way to compose it with $\texttt{GapLC}(1, \epsilon)$ to get a k-bit PCP with perfect completeness and the same soundness that of the dictatorship test. This was done in [11] and is currently the best know k-bit non-adaptive PCP with perfect completeness.

Distance from a dictator function

There are multiple notion of closeness to a dictator function. One natural definition is the minimum fraction of values we need to change such that the function becomes a dictator. There are other relaxed notions such as how close the function is to *juntas* - functions that depend on constantly many variables. Since our main motivation is the use of dictatorship test in the construction of PCP, we can work with even more relaxed notion which we describe next: For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ an influence of i^{th} variable is the probability that for a random input $x \in \{0,1\}^n$ flipping the i^{th} coordinate flips the value of the function. Note that a dictator function has a variable whose influence is 1. The influence of i^{th} variable can be expressed in terms of the fourier coefficients of f as $\mathtt{inf}_i[f] = \sum_{S \subseteq [n] | i \in S} \hat{f}(S)^2$. Using this, a degree d influence of f is $\mathtt{inf}_i^{\leq d}[f] = \sum_{S \subseteq [n] | i \in S, |S| \leq d} \hat{f}(S)^2$. We say that f is far from any dictator if for a constant d all its degree d influences are upper bounded by some small constant.

In this paper, we investigate the trade-off between the number of queries and the soundness parameter of a dictatorship test with perfect completeness w.r.t to the above defined distance to a dictator function. A random function is far from any dictator but still it passes any (non-trivial) k-query test with probability at least $1/2^k$. Thus, we cannot expect the test to have soundness parameter less than $1/2^k$. The main theorem in this paper is to show there exists a dictatorship test with perfect completeness and soundness at most $\frac{2k+1}{2^k}$.

▶ **Theorem 1.** *Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, for every k of the form $2^m - 1$ for any $m > 2$, there is a k query dictatorship test with perfect completeness and soundness $\frac{2k+1}{2^k}$.*

Our theorem improves a result of Tamaki-Yoshida[25] which had a soundness of $\frac{2k+3}{2^k}$.

▶ Remark. Tamaki-Yoshida [25] studied a k functions test where if a given set of k functions are all the same dictator then the test accepts with probability 1. They use low degree *cross influence* (Definition 2.4 in [25]) as a criteria to decide closeness to a dictator function. Our whole analysis also goes through under the same setting as that of [25], but we stick to single function version for a cleaner presentation.

1.1 Previous Work

The notion of Dictatorship Test was introduced by Bellare et al. [4] in the context of Probabilistically Checkable Proofs and also studied by Parnas et al. [20]. As our focus is on non-adpative test, for an adaptive k-bit dictatorship test, we refer interested readers to [24, 10, 9, 6]. Throughout this section, we use k to denote the number of queries and $\epsilon > 0$ an arbitrary small constant.

Getting the soundness parameter for a specific values of k had been studied earlier. For instance, for $k = 3$ Håstad [7] gave a 3-bit PCP with completeness $1-\epsilon$ and soundness $1/2+\epsilon$. It was earlier shown by Zwick [27] that any 3-bit dictator test with perfect completeness must have soundness at at least 5/8. For a 3-bit dictatorship test with perfect completeness, Khot-Saket [13] acheived a soundness parameter 20/27 and they were also able to compose their test with Label Cover towards getting 3-bit PCP with similar completeness and soundness parameters. The dictatorship test of Khot-Saket [13] was later improved by O'Donnell-Wu [17] to the optimal value of 5/8. The dictatorship test of O'Donnell-Wu [17] was used in O'Donnell-Wu [18] to get a conditional (based on Khot's d-to-1 conjecture) 3-bit PCP with perfect completeness and soundness 5/8 which was later made unconditional by Håstad [8].

For a general k, Samorodensky-Trevisan [23] constructed a k-bit PCP with imperfect completeness and soundness $2^{2\sqrt{k}}/2^k$. This was improved later by Engebretsen and Holmerin [6] to $2^{\sqrt{2k}}/2^k$ and by Håstad-Khot [9] to $2^{4\sqrt{k}}/2^k$ with perfect completeness. To break the $2^{O(\sqrt{k})}/2^k$ Samorodensky-Trevisan [24] introduced the relaxed notion of soundness (based on the low degree influences) and gave a dictatorship test (called Hypergraph dictatorship test) with almost perfect completeness and soundness $2k/2^k$ for every k and also $(k+1)/2^k$ for infinitely many k. They combined this test with Khot's Unique Games Conjecture [12] to get a conditional k-bit PCP with similar completeness and soundness guarantees. This result was improved by Austrin-Mossel [3] and they achieved $k + o(k)/2^k$ soundness.

For any k-bit CSP for which there is an instance with an integrality gap of c/s for a certain SDP, using a result of Raghavendra [21] one can get a dictatorship test with completeness $c - \epsilon$ and soundness $s + \epsilon$. Getting the explicit values of c and s for a given value of k is not clear from this result and also it cannot be used to get a dictatorship test with perfect completeness. Similarly, using the characterization of strong approximation restance of Khot et. al [14] one can get a dictatorship test but it also lacks peferct completeness. Recently, Chan [5] significantly improved the parameters for a k-bit PCP which achieves soundness $2k/2^k$ albeit losing perfect completeness. Later Huang [11] gave a k-bit PCP with perfect completeness and soundness $2^{\tilde{O}(k^{1/3})}/2^k$.

As noted earlier, the previously best known result for a k-bit dictatorship test with perfect completeness is by Tamaki-Yoshida [25]. They gave a test with soundness $\frac{2k+3}{2^k}$ for infinitely many k.

1.2 Proof Overview

Let $f : \{-1,+1\}^n \to \{-1,+1\}$ be a given balanced Boolean function [1]. Any non-adaptive k-query dictatorship test queries the function f at k locations and receives k bits which are the function output on these queries inputs. The verifier then applies some predicate, let's call it $\mathcal{P} : \{0,1\}^k \to \{0,1\}$, to the received bits and based on the outcome decides whether the function is a dictator or far from it. Since we are interested in a test with perfect completeness this puts some restriction on the set of k queried locations. If we denote $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k$ as the set of queried locations then the i^{th} bit from $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k)$ should satisfy the predicate $\mathcal{P}$. This is because, the test should always accept no matter which dictator f is.

Let μ denotes a distribution on $\mathcal{P}^{-1}(1)$. One natural way to sample $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k)$ such that the test has a perfect completeness guarantee is for each coordinate $i \in [n]$ independently sample $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k)_i$ from distribution μ. This is what we do in our dictatorship test for a specific distribution μ supported on $\mathcal{P}^{-1}(1)$. It is now easy to see that the test accepts with probability 1 of f is an i^{th} dictator for any $i \in [n]$.

Analyzing the soundness of a test is the main technical task. First note that the soundness parameter of the test depends on $\mathcal{P}^{-1}(1)$ as it can be easily verified that if f is a random function, which is far from any dictator function, then the test accepts with probability at least $\frac{|\mathcal{P}^{-1}(1)|}{2^k}$. Thus, for a better soundness guarantee we want P to have as small support

[1] Here we switch from 0/1 to +1/ − 1 for convenience. With this notation switch, balanced function means $\mathbf{E}[f(\boldsymbol{x})] = 0$

as possible. The acceptance probability of the test is given by the following expression:

$$\begin{aligned}\Pr[\text{Test accepts } f] &= \mathbf{E}[\mathcal{P}(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \cdots, f(\boldsymbol{x}_k))] \\ &= \frac{|\mathcal{P}^{-1}(1)|}{2^k} + \mathbf{E}\left[\sum_{S\subseteq[k], S\neq\emptyset} \hat{\mathcal{P}}(S) \prod_{i\in S} f(\boldsymbol{x}_i)\right]\end{aligned}$$

Thus, in order to show that the test accepts with probability at most $\frac{|\mathcal{P}^{-1}(1)|}{2^k} + \epsilon$ it is enough to show that all the expectations $E_S := |\mathbf{E}[\prod_{i\in S} f(\boldsymbol{x}_i)]|$ are small if f is far from any dictator function. Recall that at this point, we can have any predicate $\mathcal{P}$ on k bits which the verifier uses. As we will see later, for the soundness analysis we need the predicate $\mathcal{P}$ to satisfy certain properties.

For the rest of the section, assume that the given function f is such that the low degree influence of every variable $i \in [n]$ is very small constant τ. If f is a constant degree function (independent of n) then the usual analysis goes by invoking invariance principle to claim that the quantity E_S does not change by much if we replace the distribution μ to a distribution ξ over Gaussian random variable with the same first and second moments. An advantage of moving to a Gaussian distribution is that if μ was a uniform and pairwise independent distribution then so is ξ and using the fact that a pairwise independence implies a total independence in the Gaussian setting, we have $E_S \approx |\prod_{i\in S} \mathbf{E}[f(\boldsymbol{g}_i)]|$. Since we assumed that f was a balanced function we have $\mathbf{E}[f(\boldsymbol{g}_i)]| = 0$ and hence we can say that the quantity E_S is very small.

There are two main things we need to take care in the above argument. 1) We assumed that f is a low degree function and in general it may not be true. 2) The argument crucially needed μ to satisfy pairwise independence condition and hence it puts some restriction on the size of $\mathcal{P}^{-1}(1)$ (Ideally, we would like $|\mathcal{P}^{-1}(1)|$ to be as small as possible for a better soundness guarantee). We take care of (1), as in the previous works [25, 17, 3] etc., by requiring the distribution μ to have *correlation* bounded away from 1. This can be achieved by making sure the support of μ is *connected* - for every coordinate $i \in [k]$ there exists $a, b \in \mathcal{P}^{-1}(1)$ which differ at the i^{th} location. For such distribution, we can add independent *noise* to each co-ordinate without changing the quantity E_S by much. Adding independent noise has the effect that it damps the higher order fourier coefficients of f and the function behaves as a low degree function. We can now apply invariance principle to claim that $E_S \approx 0$. This was the approach in [25] and they could find a distribution μ whose support size is $2k + 3$ which is connected and pairwise independent.

In order to get an improvement in the soundness guarantee, our main technical contribution is that we can still get the overall soundness analysis to go through even if μ does not support pairwise independence condition. To this end, we start with a distribution μ whose support size is $2k + 1$ and has the property that it is *almost* pairwise independent. Since we lack pairwise independence, it introduces few obstacles in the above mentioned analysis. First, the *amount* of noise we can add to each co-ordinate has some limitations. Second, because of the limited amount of independent noise, we can no longer say that the function f behaves as a low degree function after adding the noise. With the limited amount of noise, we can say that f behaves as a low degree function as long as it does not have a large fourier mass in some interval i.e the fourier mass corresponding to $\hat{f}(T)^2$ such that $|T| \in (s, S)$ for some constant sized interval (s, S) independent of n. We handle this obstacle by designing a family of distributions $\mu_1, \mu_2, \ldots, \mu_r$ for large enough r such that the intervals that we cannot handle for different μ_i's are disjoint. Also, each μ_i has the same support and is almost pairwise independent. We then let our final test distribution as first selecting $i \in [r]$ u.a.r and then

doing the test with the corresponding distribution μ_i. Since the total fourier mass of a $-1/+1$ function is bounded by 1 and f was fixed before running the test it is very unlikely that f has a large fourier mass in the interval corresponding to the selected distribution μ_i. Hence, we can conclude that for this overall distribution, f behaves as a low degree function. We note that this approach of using family of distributions was used in [8] to construct a 3-bit PCP with perfect completeness. There it was used in the composition step.

To finish the soundness analysis, let $\tilde{f}$ be the low degree part of f. The argument in the previous paragraph concludes that $E_S \approx |\mathbf{E}[\prod_{i\in S} \tilde{f}(\boldsymbol{x}_i)]|$. As in the previous work, we can now apply invariance principle to claim that $E_S \approx |\mathbf{E}[\prod_{i\in S} \tilde{f}(\boldsymbol{g}_i)]|$ where the i^{th} coordinate $(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k)_i$ is distributed according to ξ which is almost pairwise independent. We can no longer bring the expectation inside as our distribution lacks independence. To our rescue, we have that the degree of $\tilde{f}$ is bounded by some constant independent of n. We then prove that low degree functions are robust w.r.t slight perturbation in the inputs on average. This lets us conclude $\mathbf{E}[\prod_{i\in S} \tilde{f}(\boldsymbol{g}_i)] \approx \mathbf{E}[\prod_{i\in S} \tilde{f}(\boldsymbol{h}_i)]$ where $(\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_k)_i$ is pairwise independent. We now use the property of independence of Gaussian distribution and bring the expectation inside to conclude that $E_S \approx |\mathbf{E}[\prod_{i\in S} \tilde{f}(\boldsymbol{h}_i)]| = |\prod_{i\in S} \mathbf{E}[\tilde{f}(\boldsymbol{h}_i)]| = 0$.

2 Organization

We start with some preliminaries in Section 3. In Section 4 we describe our dictatorship test. Finally, in Section 5 we prove the analysis of the described dictatorship test.

3 Preliminaries

For a positive integer k, we will denote the set $\{1, 2, \ldots, k\}$ by $[k]$. For a distribution μ, let $\mu^{\otimes n}$ denotes the n-wise product distribution.

3.1 Analysis of Boolean Function over Probability Spaces

For a function $f : \{0,1\}^n \to \mathbf{R}$, the *Fourier decomposition* of f is given by

$$f(x) = \sum_{T\subseteq[n]} \widehat{f}(T)\chi_T(x) \text{ where } \chi_T(x) := \prod_{i\in T}(-1)^{x_i} \text{ and } \widehat{f}(T) := \mathop{\mathbf{E}}_{x\in\{0,1\}^n} f(x)\chi_T(x).$$

The *Efron-Stein decomposition* is a generalization of the Fourier decomposition to product distributions of arbitrary probability spaces.

▶ **Definition 2.** Let (Ω, μ) be a probability space and $(\Omega^n, \mu^{\otimes n})$ be the corresponding product space. For a function $f : \Omega^n \to \mathbf{R}$, the Efron-Stein decomposition of f with respect to the product space is given by

$$f(x_1, \cdots, x_n) = \sum_{\beta\subseteq[n]} f_\beta(x),$$

where f_β depends only on x_i for $i \in \beta$ and for all $\beta' \not\supseteq \beta, a \in \Omega^{\beta'}$, $\mathbf{E}_{x\in\mu^{\otimes n}}[f_\beta(x) \mid x_{\beta'} = a] = 0$.

Let $\|f\|_p := \mathbf{E}_{x\in\mu^{\otimes n}}[|f(x)|^p]^{1/p}$ for $1 \le p < \infty$ and $\|f\|_\infty := \max_{x\in\Omega^{\otimes n}} |f(x)|$.

▶ **Definition 3.** For a multilinear polynomial $f : \mathbf{R}^n \to \mathbf{R}$ and any $D \in [n]$ define

$$f^{\le D} := \sum_{T\subseteq[n], |T|\le D} \hat{f}(T)\chi_T$$

i.e. $f^{\leq D}$ is degree D part of f. Also define $f^{>D} := f - f^{\leq D}$.

▶ **Definition 4.** For $i \in [n]$, the influence of the ith coordinate on f is defined as follows.

$$\mathsf{Inf}_i[f] := \mathop{\mathbf{E}}_{x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_n} \mathsf{Var}_{x_i}[f(x_1,\cdots,x_n)] = \sum_{\beta:i\in\beta} \|f_\beta\|_2^2.$$

For an integer d, the degree d influence is defined as

$$\mathsf{Inf}_i^{\leq d}[f] := \sum_{\beta:i\in\beta,|\beta|\leq d} \|f_\beta\|_2^2.$$

It is easy to see that for Boolean functions, the sum of all the degree d influences is at most d. A dictator is a function which depends on one variable. Thus, the degree 1 influence of any dictator function is 1 for some $i \in [n]$. We call a function *far* from any dictator if for every $i \in [n]$, the degree d influence is very small for some large d. This motivates the following definition.

▶ **Definition 5** ((d,τ)-quasirandom function)**.** A multilinear function $f : \mathbf{R}^n \to \mathbf{R}$ is said to be (d,τ)-quasirandom if for every $i \in [n]$ it holds that

$$\sum_{i\in S\subseteq[n],|S|\leq d} \hat{f}(S)^2 \leq \tau$$

We recall the Bonami-Beckner operator on Boolean functions.

▶ **Definition 6.** For $\gamma \in [0,1]$, the Bonami-Beckner operator $T_{1-\gamma}$ is a linear operator mapping functions $f : \{0,1\}^n \to \mathbf{R}$ to functions $T_{1-\gamma}f : \{0,1\}^n \to \mathbf{R}$ as $T_{1-\gamma}f(x) = \mathbf{E}_y[f(y)]$ where y is sampled by setting $y_i = x_i$ with probability $1-\gamma$ and y_i to be uniformly random bit with probability γ for each $i \in [n]$ independently.

We have the following relation between the fourier decomposition of $T_{1-\gamma}f$ and f.

▶ **Fact 7.** $T_{1-\gamma}f = \sum_{T\subseteq[n]}(1-\gamma)^{|T|}\hat{f}(T)\chi_T$.

4 Query efficient Dictatorship Test

We are now ready to describe our dictatorship test. The test queries a function at k locations and based on the k bits received decides if the function is a dictator or far from it. The check on the received k bits is based on a predicate with few accepting inputs which we describe next.

4.1 The Predicate

Let $k = 2^m - 1$ for some $m > 2$. Let the coordinates of the predicate is indexed by elements of $\mathbf{F}_2^m \setminus \mathbf{0} =: \{w_1, w_2, \ldots, w_{2^m-1}\}$. The Hadamard predicate H_k has following satisfying assignments:

$$H_k = \{x \in \{0,1\}^k | \exists a \in \mathbf{F}_2^m \setminus \mathbf{0} \text{ s.t } \forall i \in [k], x_i = a \cdot w_i\}$$

We will identify the set of satisfying assignments in H_k with the variables $h_1, h_2, \ldots, h_k$.

Our final predicate $\mathcal{P}_k$ is the above predicate along with few more satisfying assignments. More precisely, we add all the assignments which are at a hamming distance at most 1 from 0^k i.e. $\mathcal{P}_k = H_k \cup_{i=1}^k e_i \cup 0^k$.

4.2 The Distribution $\mathcal{D}_{k,\epsilon}$

For $0 < \epsilon \leq \frac{1}{k^2}$, consider the following distribution $\mathcal{D}_{k,\epsilon}$ on the set of satisfying assignments of $\mathcal{P}_k$ where $\alpha := (k-1)\epsilon$.

$$\begin{array}{rl} & \text{Probabilities} \qquad \text{Assignments} \\ \mathcal{D}_{k,\epsilon} \leftarrow & \{ \;\; x_1 \quad x_2 \quad \cdots\cdots \quad x_k \\ \frac{1}{1-\alpha}\left(\frac{1}{k+1}-\alpha\right) \leftarrow & \{ \;\; 0 \quad 0 \quad \cdots\cdots \quad 0 \\ \frac{1}{1-\alpha}\left(\frac{1}{k+1}-\epsilon\right) \leftarrow & \left\{ \begin{array}{c} h_1 \\ h_2 \\ \vdots \\ h_k \end{array} \right. \\ \frac{\epsilon}{1-\alpha} \leftarrow & \left\{ \begin{array}{cccc} 1 & 0 & \cdots\cdots & 0 \\ 0 & 1 & \cdots\cdots & 0 \\ & & \vdots & \\ 0 & 0 & \cdots\cdots & 1, \end{array} \right. \end{array}$$

where each h_i gets a probability mass $\frac{1}{1-\alpha}(\frac{1}{k+1}-\epsilon)$ and each e_i gets weight $\frac{\epsilon}{1-\alpha}$. The reasoning behind choosing this distribution is as follows: An uniform distribution on $H_k \cup 0^k$ has a property that it is uniform on every single co-ordinate and also pairwise independent. These two properties are very useful proving the soundness guarantee. One more property which we require is that the distribution has to be *connected*. In order to achieve this, we add k extra assignment $\{e_1, e_2, \ldots, e_k\}$ and force the distribution to be supported on all $H_k \cup_{i=1}^{k} e_i \cup 0^k$. Even though by adding extra assignments, we loose the pairwise independent property we make sure that the final distribution is *almost* pairwise independent.

We now list down the properties of this distribution which we will use in analyzing the dictatorship test. This is proved in Section B.

▶ **Observation 8.** *The distribution $\mathcal{D}_{k,\epsilon}$ above has the following properties:*

1. *$\mathcal{D}_{k,\epsilon}$ is supported on $\mathcal{P}_k$.*
2. *Marginal on every single coordinate is uniform.*
3. *For $i \neq j$, covariance of two variables x_i, x_j sampled form above distribution is: $\mathtt{Cov}[x_i, x_j] = -\frac{\epsilon}{2(1-\alpha)}$.*
4. *If we view $\mathcal{D}_{k,\epsilon}$ as a joint distribution on space $\prod_{i=1}^{k} \mathcal{X}^{(i)}$ where each $\mathcal{X}^{(i)} = \{0,1\}$, then for all $i \in [k]$, $\rho\left(\mathcal{X}^{(i)}, \prod_{j\in[k]\setminus\{i\}} \mathcal{X}^{(j)}; \mathcal{D}_{k,\epsilon}\right) \leq 1 - \frac{\epsilon^2}{2(1-\alpha)^2}$. (See Definition 14 for the definition of ρ.)*

4.3 Dictatorship Test

We will switch the notations from $\{0,1\}$ to $\{+1,-1\}$ where we identify $+1$ as 0 and -1 as 1. Let $f : \{-1,+1\}^n \to \{-1,+1\}$ be a given boolean function. We also assume that f is folded i.e. for every $\boldsymbol{x} \in \{-1,+1\}^n$, $f(\boldsymbol{x}) = -f(-\boldsymbol{x})$. We think of $\mathcal{P}_k$ as a function $\mathcal{P}_k : \{-1,+1\}^k \to \{0,1\}$ such that $P_k(z) = 1$ iff $z \in \mathcal{P}_k$. Consider the following dictatorship test:

Test $\mathcal{T}_{k,\delta}$

1. Sample $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k \in \{-1,+1\}^n$ as follows:

   a. For each $i \in [n]$, independently sample $((\boldsymbol{x}_1)_i, (\boldsymbol{x}_2)_i, \cdots, (\boldsymbol{x}_k)_i)$ according to the distribution $\mathcal{D}_{k,\delta}$.
2. Check if $(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \cdots, f(\boldsymbol{x}_k)) \in \mathcal{P}_k$.

The final test distribution is basically the above test where the parameter δ is chosen from an appropriate distribution. For a given $\frac{1}{k^2} \geq \epsilon > 0$, let $\mathsf{err} = \frac{\epsilon/5}{2^k}$ and define the following quantities : $\epsilon_0 = \epsilon$ and for $j \geq 0$, $\epsilon_{j+1} = \mathsf{err} \cdot 2^{-\left(\frac{k^{10}}{\mathsf{err}^3 \epsilon_j}\right)^k}$.

Test $\mathcal{T}'_{k,\epsilon}$

1. Set $r = \left(\frac{k}{\mathsf{err}}\right)^2$
2. Select j from $\{1, 2, \ldots, r\}$ uniformly at random.
3. Set $\delta = \epsilon_j$
4. Run test $\mathcal{T}_{k,\delta}$.

We would like to make a remark that this particular setting of ϵ_{j+1} is not very important. For our analysis, we need a sequence of ϵ_j's such that each subsequent ϵ_j is sufficiently small compared to ϵ_{j-1}.

5 Analysis of the Dictatorship Test

Notation:

We can view $f : \{-1,+1\}^n \to \{-1,+1\}$ as a function over n-fold product set $\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$ where each $\mathcal{X}_i = \{-1,+1\}^{\{i\}}$. In the test distribution $\mathcal{T}_{k,\delta}$, we can think of $\boldsymbol{x}_i$ sampled from the product distribution on $\mathcal{X}_1^{(i)} \times \mathcal{X}_2^{(i)} \times \cdots \times \mathcal{X}_n^{(i)}$. With these notations in hand, the overall distribution on $(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k)$, from the test $\mathcal{T}_{k,\delta}$, is a n-fold product distribution from the space

$$\prod_{j=1}^{n} \left(\prod_{i=1}^{k} \mathcal{X}_j^{(i)} \right).$$

where we think of $\prod_{i=1}^{k} \mathcal{X}_j^{(i)}$ as correlated space. We define the parameters for the sake of notational convenience:

1. $\beta_j := \frac{\epsilon_j}{1-(k-1)\epsilon_j}$ be the minimum probability of an atom in the distribution $\mathcal{D}_{k,\epsilon_j}$.
2. $s_{j+1} := \log(\frac{k}{\mathsf{err}})\frac{1}{\epsilon_j^2}$ and $S_j = s_{j+1}$ for $0 \leq j \leq r$.
3. $\alpha_j := (k-1)\epsilon_j$ for $j \in [r]$,

5.1 Completeness

Completeness is trivial, if f is say *ith* dictator then the test will be checking the following condition

$$((\boldsymbol{x}_1)_i, (\boldsymbol{x}_2)_i, \cdots, (\boldsymbol{x}_k)_i) \in \mathcal{P}_k$$

Using Observation 8(1), the distribution is supported on only strings in $\mathcal{P}_k$. Therefore, the test accepts with probability 1.

5.2 Soundness

▶ **Lemma 9.** *For every $\frac{1}{k^2} \geq \epsilon > 0$ there exists $0 < \tau < 1, d \in \mathbf{N}^+$ such that the following holds: Suppose f is such that for all $i \in [n]$, $\mathtt{inf}_i^{\leq d}(f) \leq \tau$, then the test $\mathcal{T}'_{k,\epsilon}$ accepts with probability at most $\frac{2k+1}{2^k} + \epsilon$. (Note: One can take τ such that $\tau^{\Omega_k(\mathsf{err}/10 s_r \log(1/\beta_r))} \leq \mathsf{err}$ and $d = \frac{\log(1/\tau)}{\log(1/\beta_r)}$.)*

Proof. The acceptance probability of the test is given by the following expression:

$$\Pr[\text{Test accepts } f] = \mathop{\mathbf{E}}_{\mathcal{T}'_{k,\epsilon}} \left[\mathcal{P}_k(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \cdots, f(\boldsymbol{x}_k))\right]$$

After expanding P_k in terms of its Fourier expansion, we get

$$\begin{aligned}
\Pr[\text{Test accepts } f] &= \frac{2k+1}{2^k} + \mathop{\mathbf{E}}_{\mathcal{T}'_{k,\epsilon}} \left[\sum_{S \subseteq [k], S \neq \emptyset} \hat{\mathcal{P}}_k(S) \prod_{i \in S} f(\boldsymbol{x}_i)\right] \\
&= \frac{2k+1}{2^k} + \sum_{S \subseteq [k], S \neq \emptyset} \hat{\mathcal{P}}_k(S) \mathop{\mathbf{E}}_{\mathcal{T}'_{k,\epsilon}} \left[\prod_{i \in S} f(\boldsymbol{x}_i)\right] \\
&\leq \frac{2k+1}{2^k} + \sum_{S \subseteq [k], S \neq \emptyset} \left| \mathop{\mathbf{E}}_{\mathcal{T}'_{k,\epsilon}} \left[\prod_{i \in S} f(\boldsymbol{x}_i)\right] \right| && (|\hat{\mathcal{P}}_k(S)| \leq 1) \\
&= \frac{2k+1}{2^k} + \sum_{S \subseteq [k], |S| \geq 2} \left| \mathop{\mathbf{E}}_{\mathcal{T}'_{k,\epsilon}} \left[\prod_{i \in S} f(\boldsymbol{x}_i)\right] \right|.
\end{aligned}$$

In the last equality, we used the fact that each $\boldsymbol{x}_i$ is distributed uniformly in $\{-1,+1\}^n$ and hence when $S = \{i\}$, $\mathbf{E}[f(\boldsymbol{x}_i)] = \hat{f}(\emptyset) = 0$. Thus, to prove the lemma it is enough to show that for all $S \subseteq [k]$ such that $|S| \geq 2$, $\mathbf{E}\left[\prod_{i \in S} f(\boldsymbol{x}_i)\right] \leq \frac{\epsilon}{2^k}$. This follows from Lemma 10. ◀

▶ **Lemma 10.** *For any $S \subseteq [k]$ such that $|S| \geq 2$,*

$$\left| \mathop{\mathbf{E}}_{j \in [r]} \left[\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{i \in S} f(\boldsymbol{x}_i)\right]\right] \right| \leq \frac{\epsilon}{2^k}$$

The proof of this follows from the following Lemmas 11 , 12, 13.

▶ **Lemma 11.** *For any $j \in [r]$ and for any $S \subseteq [k]$, $|S| \geq 2$ such that $S = \{\ell_1, \ell_2, \ldots, \ell_t\}$,*

$$\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} f(\boldsymbol{x}_{\ell_i})\right] - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{x}_{\ell_i})\right] \right| \leq 2 \cdot \mathsf{err} + k \sqrt{\sum_{s_j \leq |T| \leq S_j} \hat{f}(T)^2}.$$

where $\gamma_j = \frac{\mathsf{err}}{k s_j}$ and $d_{j,i}$ is a sequence given by $d_{j,1} = \frac{2k^2 \cdot s_j}{\mathsf{err}} \log\left(\frac{k}{\mathsf{err}}\right)$ and $d_{j,i} = (d_{j,1})^i$ for $1 < i \leq t$.

▶ **Lemma 12.** *Let $j \in [r]$ and ν_j be a distribution on jointly distributed standard Gaussian variables with same covariance matrix as that of $\mathcal{D}_{k,\epsilon_j}$. Then for any $S \subseteq [k]$, $|S| \geq 2$ such that $S = \{\ell_1, \ell_2, \ldots, \ell_t\}$,*

$$\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{x}_{\ell_i}) \right] - \mathop{\mathbf{E}}_{(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k) \sim \nu_j^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{g}_i) \right] \right| \leq \mathsf{err}_2$$

where $d_{j,i}$ from Lemma 11 and $\mathsf{err}_2 = \tau^{\Omega_k(\gamma_j/\log(1/\beta_j))}$ (Note: $\Omega(.)$ hides a constant depending on k).

▶ **Lemma 13.** *Let $k \geq 2$ and $S \subseteq [k]$ such that $|S| \geq 2$ and let $f : \mathbf{R}^n \to \mathbf{R}$ be a multilinear polynomial of degree $D \geq 1$ such that $\|f\|_2 \leq 1$. If $\mathcal{G}$ be a joint distribution on k standard gaussian random variable with a covariance matrix $(1+\delta)\ \boldsymbol{I} - \delta\ \boldsymbol{J}$ and $\mathcal{H}$ be a distribution on k independent standard gaussian then it holds that*

$$\left| \mathop{\mathbf{E}}_{\mathcal{G}^{\otimes n}} \left[\prod_{i \in S} f(\boldsymbol{g}_i) \right] - \mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} \left[\prod_{i \in S} f(\boldsymbol{h}_i) \right] \right| \leq \delta \cdot (2k)^{2kD}$$

Proofs of Lemma 11 , 12, 13 appear in Section A. We now prove Lemma 10 using the above three claims.

Proof of Lemma 10: Let $S = \{\ell_1, \ell_2, \ldots, \ell_t\}$. We are interested in getting an upper bound for the following expectation:

$$\left| \mathop{\mathbf{E}}_{j \in [r]} \left[\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} f(\boldsymbol{x}_{\ell_i}) \right] \right] \right| \leq \mathop{\mathbf{E}}_{j \in [r]} \left[\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} f(\boldsymbol{x}_{\ell_i}) \right] \right| \right].$$

Let us look at the inner expectation first. Let $\gamma_j = \frac{\mathsf{err}}{k s_j}$ and the sequence $d_{j,i}$ be from Lemma 11. We can upper bound the inner expectation as follows:

$$\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} f(\boldsymbol{x}_{\ell_i}) \right] \right| \leq \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{x}_{\ell_i}) \right] \right| + 2 \cdot \mathsf{err} + k \sqrt{\sum_{s_j \leq |T| \leq S_j} \hat{f}(T)^2}$$

(by Lemma 11)

$$\text{(by Lemma 12)} \leq \left| \mathop{\mathbf{E}}_{(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k) \sim \nu_j^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{g}_i) \right] \right| + \mathsf{err}_2 + 2 \cdot \mathsf{err} + k \sqrt{\sum_{s_j \leq |T| \leq S_j} \hat{f}(T)^2}, \tag{1}$$

where $\mathsf{err}_2 = \tau^{\Omega_k(\gamma_j/\log(1/\beta_j))}$ and ν_j has the same covariance matrix as $\mathcal{D}_{k,\epsilon_j}$. If we let $\delta_j = \frac{2\epsilon_j}{1-\alpha_j}$ then using Observation 8(3), the covariance matrix is precisely $(1+\delta_j)\ \mathbf{I} - \delta_j\ \mathbf{J}$ (note that we switched from 0/1 to $-1/+1$ which changes the covaraince by a factor of 4). Each of the functions $(T_{1-\gamma_j} f)^{\leq d_{j,i}}$ has ℓ_2 norm upper bounded by 1 and degree at most $d_{j,t}$. We can now apply Lemma 13 to conclude that

$$\left| \mathop{\mathbf{E}}_{(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k) \sim \nu_j^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{g}_i) \right] \right| \leq \left| \mathop{\mathbf{E}}_{(\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_k)} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{h}_i) \right] \right| + \delta_j \cdot (2k)^{2k d_{j,t}}, \tag{2}$$

where $\boldsymbol{h}_i$'s are independent and each $\boldsymbol{h}_i$ is distributed according to $\mathcal{N}(0,1)^n$. Thus,

$$\begin{aligned}\mathop{\mathbf{E}}_{(\boldsymbol{h}_1,\boldsymbol{h}_2,\ldots,\boldsymbol{h}_k)}\left[\prod_{\ell_i\in S}(T_{1-\gamma_j}f)^{\leq d_{j,i}}(\boldsymbol{h}_i)\right] &= \prod_{\ell_i\in S}\mathop{\mathbf{E}}_{\boldsymbol{h}_i}\left[(T_{1-\gamma_j}f)^{\leq d_{j,i}}(\boldsymbol{h}_i)\right]\\ &= \left(\widehat{(T_{1-\gamma_j}f)^{\leq d_{j,i}}}(\emptyset)\right)^t = (\hat{f}(\emptyset))^t = 0, \end{aligned} \tag{3}$$

where we used the fact that f is a folded function in the last step. Combining (1), (2) and (3), we get

$$\left|\mathop{\mathbf{E}}_{\mathcal{D}^{\otimes n}_{k,\epsilon_j}}\left[\prod_{\ell_i\in S}f(\boldsymbol{x}_{\ell_i})\right]\right| \leq \left(\delta_j\cdot(2k)^{2kd_{j,t}}\right) + \left(\tau^{\Omega_k(\gamma_j/\log(1/\beta_j))}\right) + 2\cdot\mathsf{err} + k\sqrt{\sum_{s_j\leq|T|\leq S_j}\hat{f}(T)^2} \tag{4}$$

We now upper bound the first term. For this, we use a very generous upper bounds $d_{j,1}\leq\frac{k^5}{\mathsf{err}^3}\frac{1}{\epsilon^2_{j-1}}$ and $\delta_j\leq 4\epsilon_j$.

$$\begin{aligned}\delta_j\cdot(2k)^{2kd_{j,t}} &\leq \left(4\epsilon_j\cdot(2k)^{2\mathbf{d}_{j,k}k}\right)\\ &\leq \epsilon_j\cdot 2^{\left(\frac{k^{10}}{\mathsf{err}^3\epsilon_{j-1}}\right)^k}\\ &\leq \mathsf{err}. \qquad \left(\text{using } \epsilon_j = \mathsf{err}\cdot 2^{-\left(\frac{k^{10}}{\mathsf{err}^3\epsilon_{j-1}}\right)^k}\right)\end{aligned}$$

The second term in (4) can also be upper bounded by err by choosing small enough τ.

$$\max_j\left\{\left(\tau^{\Omega_k(\gamma_j/\log(1/\beta_j))}\right)\right\} \leq \left(\tau^{\Omega_k(\gamma_r/\log(1/\beta_r))}\right) \leq \mathsf{err}.$$

Finally, taking the outer expectation of (4), we get

$$\mathop{\mathbf{E}}_{j\in[r]}\left[\left|\mathop{\mathbf{E}}_{\mathcal{D}^{\otimes n}_{k,\epsilon_j}}\left[\prod_{\ell_i\in S}f(\boldsymbol{x}_{\ell_i})\right]\right|\right] \leq 4\cdot\mathsf{err} + k\mathop{\mathbf{E}}_{j\in r}\left[\sqrt{\sum_{s_j\leq|T|\leq S_j}\hat{f}(T)^2}\right].$$

Using Cauchy-Schwartz inequality,

$$\mathop{\mathbf{E}}_{j\in[r]}\left[\sqrt{\sum_{s_j<|T|<S_j}\hat{f}(T)^2}\right] \leq \sqrt{\mathop{\mathbf{E}}_{j\in[r]}\left[\sum_{s_j<|T|<S_j}\hat{f}(T)^2\right]} \leq \frac{1}{\sqrt{r}},$$

where the last inequality uses the fact that the intervals (s_j,S_j) are disjoint for $j\in[r]$ and $\|f\|_2^2=\sum_T\hat{f}(T)^2\leq 1$. The final bound we get is

$$\left|\mathop{\mathbf{E}}_{j\in[r]}\left[\mathop{\mathbf{E}}_{\mathcal{D}^{\otimes n}_{k,\epsilon_j}}\left[\prod_{\ell_i\in S}f(\boldsymbol{x}_{\ell_i})\right]\right]\right| \leq \mathop{\mathbf{E}}_{j\in[r]}\left[\left|\mathop{\mathbf{E}}_{\mathcal{D}^{\otimes n}_{k,\epsilon_j}}\left[\prod_{\ell_i\in S}f(\boldsymbol{x}_{\ell_i})\right]\right|\right] \leq 4\cdot\mathsf{err}+\frac{k}{\sqrt{r}} \leq 5.\mathsf{err} \leq \frac{\epsilon}{2^k},$$

as required. ◀

References

1 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. doi:10.1145/278298.278306.

2 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. doi:10.1145/273865.273901.

3 Per Austrin and Elchanan Mossel. Approximation resistant predicates from pairwise independence. *Computational Complexity*, 18(2):249–271, 2009. doi:10.1007/s00037-009-0272-6.

4 Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free Bits, PCPs, and Nonapproximability—Towards Tight Results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

5 Siu On Chan. Approximation Resistance from Pairwise Independent Subgroups. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 447–456. ACM, 2013.

6 Lars Engebretsen and Jonas Holmerin. More efficient queries in PCPs for NP and improved approximation hardness of maximum CSP. *Random Structures & Algorithms*, 33(4):497–514, 2008.

7 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

8 Johan Håstad. On the NP-hardness of Max-Not-2. *SIAM Journal on Computing*, 43(1):179–193, 2014.

9 Johan Håstad and Subhash Khot. Query Efficient PCPs with Perfect Completeness. *Theory of Computing*, 1(1):119–148, 2005.

10 Johan Håstad and Avi Wigderson. Simple analysis of graph tests for linearity and PCP. *Random Structures & Algorithms*, 22(2):139–160, 2003.

11 Sangxia Huang. Approximation resistance on satisfiable instances for predicates with few accepting inputs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 457–466. ACM, 2013.

12 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775. ACM, 2002.

13 Subhash Khot and Rishi Saket. A 3-query non-adaptive PCP with perfect completeness. In *21st Annual IEEE Conference on Computational Complexity (CCC'06)*, pages 11–pp. IEEE, 2006.

14 Subhash Khot, Madhur Tulsiani, and Pratik Worah. A characterization of strong approximation resistance. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 634–643. ACM, 2014.

15 Elchanan Mossel. Gaussian bounds for noise correlation of functions. *Geom. Funct. Anal.*, 19(6):1713–1756, 2010. (Preliminary version in *49th FOCS*, 2008). doi:10.1007/s00039-010-0047-x.

16 Elchanan Mossel, Ryan O'Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 21–30. IEEE, 2005.

17 Ryan O'Donnell and Yi Wu. 3-bit dictator testing: 1 vs. 5/8. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 365–373. Society for Industrial and Applied Mathematics, 2009.

18 Ryan O'Donnell and Yi Wu. Conditional hardness for satisfiable 3-CSPs. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 493–502. ACM, 2009.

19 Krzysztof Oleszkiewicz. On a nonsymmetric version of the khinchine-kahane inequality. In *Stochastic inequalities and applications*, pages 157–168. Springer, 2003.

20 Michal Parnas, Dana Ron, and Alex Samorodnitsky. Testing basic Boolean Formulae. *SIAM Journal on Discrete Mathematics*, 16(1):20–46, 2002.

21 Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 245–254. ACM, 2008.

22 Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998. doi:10.1137/S0097539795280895.

23 Alex Samorodnitsky and Luca Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 191–199. ACM, 2000.

24 Alex Samorodnitsky and Luca Trevisan. Gowers uniformity, influence of variables, and PCPs. *SIAM Journal on Computing*, 39(1):323–360, 2009.

25 Suguru Tamaki and Yuichi Yoshida. A query efficient non-adaptive long code test with perfect completeness. *Random Structures & Algorithms*, 47(2):386–406, 2015.

26 Pawel Wolff. Hypercontractivity of simple random variables. *Studia Mathematica*, 180(3):219–236, 2007.

27 Uri Zwick. Approximation Algorithms for Constraint Satisfaction Problems Involving at Most Three Variables per Constraint. In *In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997.

A Proofs of Lemma 11, 12 & 13

In this section, we provide proofs of three crucial lemmas which we used in proving the soundness analysis of our dictatorship test. We start with some more preliminaries.

A.1 Correlated Spaces

Let $\Omega_1 \times \Omega_2$ be two correlated spaces and μ denotes the joint distribution. Let μ_1 and μ_2 denote the marginal of μ on space Ω_1 and Ω_2 respectively. The correlated space $\rho(\Omega_1 \times \Omega_2; \mu)$ can be represented as a bipartite graph on (Ω_1, Ω_2) where $x \in \Omega_1$ is connected to $y \in \Omega_2$ iff $\mu(x, y) > 0$. We say that the correlated spaces is *connected* if this underlying graph is connected.

We need a few definitions and lemmas related to correlated spaces defined by Mossel [15].

▶ **Definition 14.** Let $(\Omega_1 \times \Omega_2, \mu)$ be a finite correlated space, the correlation between Ω_1 and Ω_2 with respect to μ us defined as

$$\rho(\Omega_1, \Omega_2; \mu) := \max_{\substack{f:\Omega_1 \to \mathbf{R}, \mathbf{E}[f]=0, \mathbf{E}[f^2] \le 1 \\ g:\Omega_2 \to \mathbf{R}, \mathbf{E}[g]=0, \mathbf{E}[g^2] \le 1}} \mathop{\mathbf{E}}_{(x,y)\sim\mu} [|f(x)g(y)|].$$

The following result (from [15]) provides a way to upper bound correlation of a correlated spaces.

▶ **Lemma 15.** *Let $(\Omega_1 \times \Omega_2, \mu)$ be a finite correlated space such that the probability of the smallest atom in $\Omega_1 \times \Omega_2$ is at least $\alpha > 0$ and the correlated space is connected then*

$$\rho(\Omega_1, \Omega_2; \mu) \le 1 - \alpha^2/2$$

▶ **Definition 16** (Markov Operator)**.** Let $(\Omega_1 \times \Omega_2, \mu)$ be a finite correlated space, the Markov operator, associated with this space, denoted by U, maps a function $g : \Omega_2 \to \mathbf{R}$ to functions $Ug : \Omega_1 \to \mathbf{R}$ by the following map:

$$(Ug)(x) := \mathop{\mathbf{E}}_{(X,Y)\sim\mu} [g(Y) \mid X = x].$$

In the soundness analysis of our dictatorship test, we will need to understand the Efron-Stein decomposition of Ug in terms of the decomposition of g. The following proposition gives a way to relate these two decompositions.

▶ **Proposition 17** ([15, Proposition 2.11]). *Let $(\prod_{i=1}^n \Omega_i^{(1)} \times \prod_{i=1}^n \Omega_i^{(2)}, \prod_{i=1}^n \mu_i)$ be a product correlated spaces. Let $g : \prod_{i=1}^n \Omega_i^{(2)} \to \mathbf{R}$ be a function and U be the Markov operator mapping functions form space $\prod_{i=1}^n \Omega_i^{(2)}$ to the functions on space $\prod_{i=1}^n \Omega_i^{(1)}$. If $g = \sum_{S\subseteq[n]} g_S$ and $Ug = \sum_{S\subseteq[n]} (Ug)_S$ be the Efron-Stein decomposition of g and Ug respectively then,*

$$(Ug)_S = U(g_S)$$

i.e. the Efron-Stein decomposition commutes with Markov operators.

Finally, the following proposition says that if the correlation between two spaces is bounded away from 1 then *higher order* terms in the Efron-Stein decomposition of Ug has a very small ℓ_2 norm compared to the ℓ_2 norm of the corresponding higher order terms in the Efron-Stein decomposition of g.

▶ **Proposition 18** ([15, Proposition 2.12]). *Assume the setting of Proposition 17 and furthermore assume that $\rho(\Omega_i^{(1)}, \Omega_i^{(2)}; \mu_i) \leq \rho$ for all $i \in [n]$, then for all g it holds that*

$$\|U(g_S)\|_2 \leq \rho^{|S|}\|g_S\|_2.$$

A.2 Hypercontractivity

▶ **Definition 19.** A random variable r is said to be (p, q, η)-hypercontractive if it satisfies

$$\|a + \eta r\|_q \leq \|a + r\|_p$$

for all $a \in \mathbf{R}$.

We note down the hypercontractive parameters for Rademacher random variable (uniform over ± 1) and standard gaussian random variable.

▶ **Theorem 20** ([26][19]). *Let X denote either a uniformly random ± 1 bit, a standard one-dimensional Gaussian. Then X is $\left(2, q, \frac{1}{\sqrt{q-1}}\right)$-hypercontractive.*

The following proposition says that the higher norm of a low degree function w.r.t hypercontractive sequence of ensembles is bounded above by its second norm.

▶ **Proposition 21** ([16]). *Let $\boldsymbol{x}$ be a $(2, q, \eta)$-hypercontractive sequence of ensembles and Q be a multilinear polynomial of degree d. Then*

$$\|Q(\boldsymbol{x})\|_q \leq \eta^{-d}\|Q(\boldsymbol{x})\|_2$$

A.3 Invariance Principle

Let μ be any distribution on $\{-1, +1\}^k$. Consider the following distribution on $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k \in \{-1, +1\}^n$ such that independently for each $i \in [n]$, $((\boldsymbol{x}_1)_i, (\boldsymbol{x}_2)_i, \ldots, (\boldsymbol{x}_k)_i)$ is sampled from μ. We will denote this distribution as $\mu^{\otimes n}$. We are interested in evaluation of a multilinear polynomial $f : \mathbf{R}^n \to \mathbf{R}$ on $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k)$ sampled as above.
Invariance principle shows the closeness between two different distributions w.r.t some quantity of interest. We are now ready to state the version of the invariance principle from [15] that we need.

▶ **Theorem 22** ([15]). *For any $\alpha > 0, \epsilon > 0, k \in \mathbf{N}^+$ there are $d, \tau > 0$ such that the following holds: Let μ be the distribution on $\{+1, -1\}^k$ satisfying*
1. $\mathbf{E}_{x\sim\mu}[x_i] = 0$ *for every* $i \in [k]$
2. $\mu(x) \geq \alpha$ *for every* $x \in \{-1, +1\}^k$ *such that* $\mu(x) \neq 0$

Let ν be a distribution on standard jointly distributed Gaussian variables with the same covariance matrix as distribution μ. Then, for every set of k (d, τ)-quasirandom multilinear polynomials $f_i : \mathbf{R}^n \to \mathbf{R}$, and suppose $\mathsf{Var}[f_i^{>d}] \leq (1-\gamma)^{2d}$ for $0 < \gamma < 1$ it holds that

$$\left| \mathop{\mathbf{E}}_{(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k) \sim \mu^{\otimes n}} \left[\prod_{i=1}^{k} f_i(\boldsymbol{x}_i) \right] - \mathop{\mathbf{E}}_{(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k) \sim \nu^{\otimes n}} \left[\prod_{i=1}^{k} f_i(\boldsymbol{g}_i) \right] \right| \leq \epsilon$$

(Note: one can take $d = \frac{\log(1/\tau)}{\log(1/\alpha)}$ and τ such that $\epsilon = \tau^{\Omega(\gamma/\log(1/\alpha))}$, where $\Omega(.)$ hides constant depending only on k.)

A.4 Moving to a low degree function

The following lemma, at a very high level, says that if change f to its low degree *noisy version* then the loss we incur in the expected quantity is small.

▶ **Lemma 23** (Restatement of Lemma 11). *For any $j \in [r]$ and for any $S \subseteq [k]$, $|S| \geq 2$ such that $S = \{\ell_1, \ell_2, \ldots, \ell_t\}$,*

$$\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} f(\boldsymbol{x}_{\ell_i}) \right] - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{x}_{\ell_i}) \right] \right| \leq 2 \cdot \mathsf{err} + k \sqrt{\sum_{s_j \leq |T| \leq S_j} \hat{f}(T)^2}.$$

where $\gamma_j = \frac{\mathsf{err}}{k s_j}$ and $d_{j,i}$ is a sequence given by $d_{j,1} = \frac{2k^2 \cdot s_j}{\mathsf{err}} \log\left(\frac{k}{\mathsf{err}}\right)$ and $d_{j,i} = (d_{j,1})^i$ for $1 < i \leq t$.

Proof. The proof is presented in two parts. We first prove an upper bound on

$$\Gamma_1 := \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} f(\boldsymbol{x}_{\ell_i}) \right] - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] \right| \leq \mathsf{err} + k \sqrt{\sum_{s_j \leq |T| \leq S_j} \hat{f}(T)^2} \tag{5}$$

and then an upper bound on

$$\Gamma_2 := \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{\ell_i \in S} (T_{1-\gamma_j} f)^{\leq d_{j,i}}(\boldsymbol{x}_{\ell_i}) \right] \right| \leq \mathsf{err}. \tag{6}$$

Note that both these upper bounds are enough to prove the lemma.

Upper Bounding Γ_1: The following analysis is very similar to the one in [25], we reproduce it here for the sake of completeness. The first upper bound is obtained by getting the upper bound for the following, for every $a \in [t]$.

$$\Gamma_{1,a} := \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{i \geq a} f(\boldsymbol{x}_{\ell_i}) \prod_{i < a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{i > a} f(\boldsymbol{x}_{\ell_i}) \prod_{i \leq a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] \right| \tag{7}$$

Note that by triangle inequality, $\Gamma_1 \leq \sum_{a\in[t]} \Gamma_{1,a}$.

$$
\begin{aligned}
(7) &= \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\left(f(\boldsymbol{x}_{\ell_a}) - T_{1-\gamma_j} f(\boldsymbol{x}_{\ell_a})\right) \prod_{i>a} f(\boldsymbol{x}_{\ell_i}) \prod_{i<a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] \right| \\
&= \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\left(id - T_{1-\gamma_j}\right) f(\boldsymbol{x}_{\ell_a}) \prod_{i>a} f(\boldsymbol{x}_{\ell_i}) \prod_{i<a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] \right| \\
&= \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[U\left(\left(id - T_{1-\gamma_j}\right) f\right)(\boldsymbol{x}_{\{\ell_i : i\in[t]\setminus\{a\}\}}) \prod_{i>a} f(\boldsymbol{x}_{\ell_i}) \prod_{i<a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \right] \right| \qquad (8)
\end{aligned}
$$

where U is the Markov operator for the correlated probability space which maps functions from the space $\mathcal{X}^{(\ell_a)}$ to the space $\prod_{i\in[t]\setminus\{a\}} \mathcal{X}^{(\ell_i)}$. We can look at the above expression as a product of two functions, $F = \prod_{i>a} f \prod_{i<a} (T_{1-\gamma_j} f)$ and $G = U(id - T_{1-\gamma_j})f)$. From Observation 8(4), the correlation between spaces $\left(\mathcal{X}^{(\ell_a)}, \prod_{i\in[t]\setminus\{a\}} \mathcal{X}^{(\ell_i)}\right)$ is upper bounded by $1 - \left(\frac{\epsilon_j}{1-\alpha_j}\right)^2 \leq 1 - \epsilon_j^2 =: \rho_j$. Taking the Efron-Stein decomposition with respect to the product distribution, we have the following because of orthogonality of the Efron-Stein decomposition,

$$
\begin{aligned}
(8) &= \left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} [G \times F] \right| = \left| \sum_{T\subseteq[n]} \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} [G_T \times F_T] \right| \\
\text{(by Cauchy-Schwartz)} \quad &\leq \sqrt{\sum_{T\subseteq[n]} \|F_T\|_2^2} \sqrt{\sum_{T\subseteq[n]} \|G_T\|_2^2} \qquad (9)
\end{aligned}
$$

where the norms are with respect to $\mathcal{D}_{k,\epsilon_j}^{\otimes n}$'s marginal distribution on the product distribution $\prod_{i\in[t]\setminus\{a\}} \mathcal{X}^{(\ell_i)}$. By orthogonality, the quantity $\sqrt{\sum_{T\subseteq[n]} \|F_T\|_2^2}$ is just $\|F\|_2$. As F is product of function whose range is $[-1,+1]$, rane of F is also $[-1,+1]$ and hence $\|F\|_2$ is at most 1. Therefore,

$$
(9) \leq \sqrt{\sum_{T\subseteq[n]} \|G_T\|_2^2} \qquad (10)
$$

We have $G_T = (UG')_T$, where $G' = (id - T_{1-\gamma_j})f$. In G'_T, the Efron-Stein decomposition is with respect to the marginal distribution of $\mathcal{D}_{k,\epsilon_j}^{\otimes n}$ on $\mathcal{X}^{(\ell_a)}$, which is just uniform (by Observation 8(2)). Using Proposition 17, we have $G_T = UG'_T = U(id - T_{1-\gamma_j})f_T$. Substituting in (10), we get

$$
(10) = \sqrt{\sum_{T\subseteq[n]} \|U(if - T_{1-\gamma_j})f_T)\|_2^2} \qquad (11)
$$

We also have that the correlation is upper bounded by ρ_j. We can therefore apply Proposition 18, and conclude that for each $T \subseteq [n]$,

$$
\|U(id - T_{1-\gamma_j})f_T\|_2 \leq \rho_j^{|T|} \|(id - T_{1-\gamma_j})f_T\|_2
$$

where the norm on the right is with respect to the uniform distribution. Observe that

$$
\|(id - T_{1-\gamma_j})f_T\|_2^2 = (1 - (1-\gamma_j)^{|T|})^2 \hat{f}(T)^2
$$

Substituting back into (11), we get

$$(11) \leq \sqrt{\sum_{T\subseteq[n]} \underbrace{\rho_j^{2|T|}(1-(1-\gamma_j)^{|T|})^2\hat{f}(T)^2}_{\mathsf{Term}(\epsilon_j,\gamma_j,T)}} \tag{12}$$

We will now break the above summation into three different parts and bound each part separately.

$$\Theta_1 := \sum_{\substack{T\subseteq[n],\\ |T|\leq s_j}} \mathsf{Term}(\epsilon_j,\gamma_j,T) \qquad \Theta_2 := \sum_{\substack{T\subseteq[n],\\ s_j<|T|<S_j}} \mathsf{Term}(\epsilon_j,\gamma_j,T)$$

$$\Theta_3 := \sum_{\substack{T\subseteq[n],\\ |T|\geq S_j}} \mathsf{Term}(\epsilon_j,\gamma_j,T)$$

- **Upper bounding Θ_1:**

$$\Theta_1 = \sum_{\substack{T\subseteq[n],\\ |T|\leq s_j}} \mathsf{Term}(\epsilon_j,\gamma_j,T) = \sum_{\substack{T\subseteq[n],\\ |T|\leq s_j}} \rho_j^{2|T|}(1-(1-\gamma_j)^{|T|})^2\hat{f}(T)^2$$
$$\leq \sum_{\substack{T\subseteq[n],\\ |T|\leq s_j}} (1-(1-\gamma_j)^{|T|})^2\hat{f}(T)^2.$$

  For every $|T| \leq s_j$ we have $1-(1-\gamma_j)^{|T|} \leq \mathsf{err}_1/k$. Thus,

$$\Theta_1 \leq \left(\frac{\mathsf{err}_1}{k}\right)^2 \sum_{\substack{T\subseteq[n],\\ |T|\leq s_j}} \hat{f}(T)^2.$$

- **Upper bounding Θ_3:**

$$\Theta_3 = \sum_{\substack{T\subseteq[n],\\ |T|\geq S_j}} \mathsf{Term}(\epsilon_j,\gamma_j,T) = \sum_{\substack{T\subseteq[n],\\ |T|\geq S_j}} \rho_j^{2|T|}(1-(1-\gamma_j)^{|T|})^2\hat{f}(T)^2 \leq \sum_{\substack{T\subseteq[n],\\ |T|\geq S_j}} \rho_j^{2|T|}\hat{f}(T)^2.$$

  For every $|T| \geq S_j$ we have $\rho_j^{|T|} \leq (1-\epsilon_j^2)^{|T|} \leq \mathsf{err}_1/k$. Thus,

$$\Theta_3 \leq \left(\frac{\mathsf{err}_1}{k}\right)^2 \sum_{\substack{T\subseteq[n],\\ |T|\geq S_j}} \hat{f}(T)^2.$$

Substituting these upper bounds in (12),

$$\Gamma_{1,a} \leq \sqrt{\left(\frac{\mathsf{err}_1}{k}\right)^2 \sum_{\substack{T\subseteq[n],\\ |T|\leq s_j \text{ or } |T|\geq S_j}} \hat{f}(T)^2 + \sum_{\substack{T\subseteq[n],\\ s_j<|T|<S_j}} \hat{f}(T)^2}$$
$$\leq \sqrt{\left(\frac{\mathsf{err}_1}{k}\right)^2 + \sum_{s_j<|T|<S_j} \hat{f}(T)^2} \qquad \text{(since } \textstyle\sum_T \hat{f}(T)^2 \leq 1\text{)}$$
$$\leq \frac{\mathsf{err}_1}{k} + \sqrt{\sum_{s_j<|T|<S_j} \hat{f}(T)^2}. \qquad \text{(using concavity)}$$

The required upper bound on Γ_1 follows by using $\Gamma_1 \leq \sum_{a\in[t]} \Gamma_{1,a}$ and the above bound.

Upper Bounding Γ_2 : We will now show an upper bound on Γ_2. The approach is similar to the previous case, we upper bound the following quantity for every $a \in [t]$

$$\begin{aligned}
\Gamma_{2,a} := &\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{i \geq a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \prod_{i<a} (T_{1-\gamma_j} f^{\leq d_{j,i}})(\boldsymbol{x}_{\ell_i}) \right] \right. \\
&\left. - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\prod_{i > a} (T_{1-\gamma_j} f)(\boldsymbol{x}_{\ell_i}) \prod_{i \leq a} (T_{1-\gamma_j} f^{\leq d_{j,i}})(\boldsymbol{x}_{\ell_i}) \right] \right| \\
= &\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\left(T_{1-\gamma_j} f(\boldsymbol{x}_{\ell_a}) - T_{1-\gamma_j} f^{\leq d_{j,a}}(\boldsymbol{x}_{\ell_a})\right) \prod_{i>a} T_{1-\gamma_j} f(\boldsymbol{x}_{\ell_i}) \prod_{i<a} (T_{1-\gamma_j} f^{\leq d_{j,i}})(\boldsymbol{x}_{\ell_i}) \right] \right| \\
= &\left| \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}} \left[\left(T_{1-\gamma_j} f^{> d_{j,a}}(\boldsymbol{x}_{\ell_a})\right) \prod_{i>a} T_{1-\gamma_j} f(\boldsymbol{x}_{\ell_i}) \prod_{i<a} (T_{1-\gamma_j} f^{\leq d_{j,i}})(\boldsymbol{x}_{\ell_i}) \right] \right| \qquad (13)
\end{aligned}$$

By using Holder's inequality we can upper bound (13) as:

$$(13) \leq \|T_{1-\gamma_j} f^{>d_{j,a}}\|_2 \prod_{i>a} \|T_{1-\gamma_j} f\|_{2(t-1)} \prod_{i<a} \|T_{1-\gamma_j} f^{\leq d_{j,i}}\|_{2(t-1)}, \qquad (14)$$

where each norm is w.r.t the uniform distribution as marginal of each $\boldsymbol{x}_{\ell_i}$ is uniform in $\{+1,-1\}^n$. Now, $\|T_{1-\gamma_j} f\|_{2(t-1)} \leq 1$ as the range if $T_{1-\gamma_j} f$ is in $[-1,+1]$. To upper bound $\|T_{1-\gamma_j} f^{\leq d_{j,i}}\|_{2(t-1)}$, we use Proposition 21 and using the fact that $\{-1,+1\}$ uniform random variable is $(2, q, 1/\sqrt{q-1})$ hypercontractive (Theorem 20) to get

$$\|T_{1-\gamma_j} f^{\leq d_{j,i}}\|_{2(t-1)} \leq (2t-3)^{d_{j,i}} \|T_{1-\gamma_j} f^{\leq d_{j,i}}\|_2 \leq (2t)^{d_{j,i}}.$$

Plugging this in (14), we get

$$\begin{aligned}
(14) &\leq \|T_{1-\gamma_j} f^{>d_{j,a}}\|_2 \prod_{i<a} (2t)^{d_{j,i}} \leq (1-\gamma_j)^{d_{j,a}} \cdot \prod_{i<a} (2t)^{d_{j,i}} \\
&\leq e^{-\gamma_j d_{j,a}} \cdot (2k)^{k \cdot d_{j,a-1}} \\
&\leq e^{-\frac{\mathsf{err}}{k s_j} \cdot d_{j,a}} \cdot (2k)^{k \cdot d_{j,a-1}} \qquad (15)
\end{aligned}$$

Now,

$$\begin{aligned}
d_{j,1} \cdot d_{j,a-1} &= d_{j,a} \\
\frac{2k^2 \cdot s_j}{\mathsf{err}} \log\left(\frac{k}{\mathsf{err}}\right) \cdot d_{j,a-1} &= d_{j,a} \\
\frac{k^2 \cdot s_j}{\mathsf{err}} \log\left(\frac{k}{\mathsf{err}}\right) + \frac{k^2 \cdot s_j}{\mathsf{err}} \log\left(\frac{k}{\mathsf{err}}\right) \cdot d_{j,a-1} &\leq d_{j,a} \\
\frac{k \cdot s_j}{\mathsf{err}} \log\left(\frac{k}{\mathsf{err}}\right) + \frac{k^2 \cdot s_j}{\mathsf{err}} \cdot \log(2k) \cdot d_{j,a-1} &\leq d_{j,a} \\
\frac{k \cdot s_j}{\mathsf{err}} \cdot \left(\log\left(\frac{k}{\mathsf{err}}\right) + k \cdot d_{j,a-1} \log(2k)\right) &= d_{j,a} \\
\frac{k \cdot s_j}{\mathsf{err}} \cdot \log\left(\frac{k}{\mathsf{err}} (2k)^{k \cdot d_{j,a-1}}\right) &= d_{j,a}
\end{aligned}$$

This implies

$$\log\left(\frac{k}{\mathsf{err}}(2k)^{k\cdot d_{j,a-1}}\right) = \frac{\mathsf{err}}{ks_j}\cdot d_{j,a}$$
$$\implies \frac{k}{\mathsf{err}}(2k)^{k\cdot d_{j,a-1}} = e^{\frac{\mathsf{err}}{ks_j}\cdot d_{j,a}}$$
$$\implies e^{-\frac{\mathsf{err}}{ks_j}\cdot d_{j,a}}\cdot(2k)^{k\cdot d_{j,a-1}} = \frac{\mathsf{err}}{k}.$$

Thus from (15), we have $\Gamma_{2,a} \le \frac{\mathsf{err}}{k}$. To conclude the proof, by triangle inequality we have $\Gamma_2 \le \sum_{a\in[t]}\Gamma_{2,a} \le \mathsf{err}$. ◀

A.5 Moving to the Gaussian setting

We are now in the setting of *low degree* polynomials because of Lemma 11. The following lemma let us switch from our test distribution to a Gaussian distribution with the same first two moments.

▶ **Lemma 24** (Restatement of Lemma 12)**.** *Let $j \in [r]$ and ν_j be a distribution on jointly distributed standard Gaussian variables with same covariance matrix as that of $\mathcal{D}_{k,\epsilon_j}$. Then for any $S \subseteq [k]$, $|S| \ge 2$ such that $S = \{\ell_1, \ell_2, \ldots, \ell_t\}$,*

$$\left|\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon_j}^{\otimes n}}\left[\prod_{\ell_i\in S}(T_{1-\gamma_j}f)^{\le d_{j,i}}(\boldsymbol{x}_{\ell_i})\right] - \mathop{\mathbf{E}}_{(\boldsymbol{g}_1,\boldsymbol{g}_2,\ldots,\boldsymbol{g}_k)\sim\nu_j^{\otimes n}}\left[\prod_{\ell_i\in S}(T_{1-\gamma_j}f)^{\le d_{j,i}}(\boldsymbol{g}_i)\right]\right| \le \mathsf{err}_2$$

where $d_{j,i}$ from Lemma 11 and $\mathsf{err}_2 = \tau^{\Omega_k(\gamma_j/\log(1/\beta_j))}$ (Note: $\Omega(.)$ hides a constant depending on k).

Proof. Using the definition of (d,τ)-quasirandom function and Fact 7, if f is (d,τ)- quasirandom then so is $T_{1-\gamma}f$ for any $0 \le \gamma \le 1$. Also, $T_{1-\gamma}f$ satisfies

$$\mathsf{Var}[T_{1-\gamma}f^{>d}] = \sum_{\substack{T\subseteq[n]\\|T|>d}}(1-\gamma)^{2|T|}\hat{f}(T)^2 \le (1-\gamma)^{2d}\cdot\sum_{\substack{T\subseteq[n]\\|T|>d}}\hat{f}(T)^2 \le (1-\gamma)^{2d}.$$

The lemma follows from a direct application of Theorem 22. ◀

A.6 Making Gaussian variables independent

Our final lemma allows us to make the Gaussian variables independent. Here we crucially need the property that the polynomials we are dealing with are low degree polynomials. Before proving Lemma 13, we need the following lemma which says that low degree functions are robust to small perturbations in the input on average.

▶ **Lemma 25.** *Let $f : \mathbf{R}^n \to \mathbf{R}$ be a multilinear polynomial of degree d such that $\|f\|_2 \le 1$ suppose $\boldsymbol{x}, \boldsymbol{z} \sim \mathcal{N}(0,1)^n$ be n-dimensional standard gaussian vectors such that $\mathbf{E}[x_i z_i] \ge 1-\delta$ for all $i \in [n]$. Then*

$$\mathbf{E}[(f(\boldsymbol{x}) - f(\boldsymbol{z}))^2] \le 2\delta d.$$

Proof. For $T \subseteq [n]$, we have

$$\mathbf{E}[\chi_T(\boldsymbol{x})\chi_T(\boldsymbol{z})] = \prod_{i\in T}\mathbf{E}[x_i z_i] \ge \prod_{i\in T}(1-\delta) \ge (1-\delta)^{|T|}$$

We now bound the following expression,

$$\begin{aligned}\mathbf{E}[(f(\boldsymbol{x})-f(\boldsymbol{z}))^2] &= \mathbf{E}[f(\boldsymbol{x})^2+f(\boldsymbol{z})^2-2f(\boldsymbol{x})z(\boldsymbol{x})]\\ &= \sum_{T\subseteq[n],|T|\le d}\hat{f}(T)^2(2-2\,\mathbf{E}[\chi_T(\boldsymbol{x})\chi_T(\boldsymbol{z})])\\ &\le 2\cdot\sum_{T\subseteq[n],|T|\le d}\hat{f}(T)^2(1-(1-\delta)^{|T|})\\ &\le 2\cdot\sum_{T\subseteq[n],|T|\le d}\hat{f}(T)^2\delta|T|\\ &\le 2\delta d\cdot\sum_{T\subseteq[n],|T|\le d}\hat{f}(T)^2\le 2\delta d,\end{aligned}$$

where the last inequality uses $\|f\|_2\le 1$. ◀

We are now ready to prove Lemma 13.

▶ **Lemma 26** (Restatement of Lemma 13). *Let $k\ge 2$ and $2\le t\le k$ and let $f:\mathbf{R}^n\to\mathbf{R}$ be a multilinear polynomial of degree $D\ge 1$ such that $\|f\|_2\le 1$. If $\mathcal{G}$ be a joint distribution on k standard gaussian random variable with covariance matrix $(1+\delta)\ \boldsymbol{I}-\delta\ \boldsymbol{J}$ and $\mathcal{H}$ be a distribution on k independent standard gaussian then it holds that*

$$\left|\mathop{\mathbf{E}}_{\mathcal{G}^{\otimes n}}\left[\prod_{i\in[t]}f(\boldsymbol{g}_i)\right]-\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}}\left[\prod_{i\in[t]}f(\boldsymbol{h}_i)\right]\right|\le\delta\cdot(2k)^{2Dk}.$$

Proof. Let $\mathbf{\Sigma}=(1+\delta)\ \mathbf{I}-\delta\ \mathbf{J}$ be the covariance matrix. Let $\mathbf{M}=(1-\delta')((1+\beta)\ \mathbf{I}-\beta\ \mathbf{J})$ be a matrix such that $\mathbf{M^2}=\mathbf{\Sigma}$. There are multiple $\mathbf{M}$ which satisfy $\mathbf{M^2}=\mathbf{\Sigma}$. We chose the $\mathbf{M}$ stated above to make the analysis simpler. From the way we chose $\mathbf{M}$ and using the condition $\mathbf{M^2}=\mathbf{\Sigma}$, it is easy to observe that β and δ' should satisfy the following two conditions:

$$1-\delta'=\frac{1}{\sqrt{1+(k-1)\beta^2}}\quad\text{and}\quad\frac{(k-2)\beta^2-2\beta}{1+(k-1)\beta^2}=-\delta.$$

Since $\mathcal{H}$ is a distribution of k independent standard gaussians, we can generate a sample $x\sim\mathcal{G}$ by sampling $y\sim\mathcal{H}$ and setting $x=\ \mathbf{M}y$. In what follows, we stick to the following notation: $(\boldsymbol{h}_1,\boldsymbol{h}_2,\ldots,\boldsymbol{h}_k)\sim\mathcal{H}^{\otimes n}$ and $(\boldsymbol{g}_1,\boldsymbol{g}_2,\ldots,\boldsymbol{g}_k)_j=\ \mathbf{M}(\boldsymbol{h}_1,\boldsymbol{h}_2,\ldots,\boldsymbol{h}_k)_j$ for each $j\in[n]$.

Because of the way we chose to generate $g_i's$, we have for all $i\in[k]$ and $j\in[n]$, $\mathbf{E}[(\boldsymbol{g}_i)_j(\boldsymbol{h}_i)_j]=1-\delta'\ge 1-k\beta^2$. To get an upper bound on β, notice that β is a root of the quadratic equation $(k+\delta k-\delta-2)\beta^2-2\beta+\delta=0$. Let $k'=(k+\delta k-\delta-2)$, if β_1,β_2 are the roots of the equation then they satisfy: $k'\beta_1+k'\beta_2=2$ and $(k'\beta_1)(k'\beta_2)=\delta k'$ and $\beta_1,\beta_2>0$. Thus, we have $\min\{k'\beta_1,k'\beta_2\}\le\delta k'$ and hence, we can take β such that $\beta\le\delta$.

We wish to upper bound the following expression:

$$\Gamma:=\left|\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}}\left[\prod_{i\in[t]}f(\boldsymbol{g}_i)-\prod_{i\in[t]}f(\boldsymbol{h}_i)\right]\right|.$$

Define the following quantity

$$\Gamma_i:=\left|\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}}\left[\prod_{j=1}^{i-1}f(\boldsymbol{h}_j)\prod_{j=i}^{t}f(\boldsymbol{g}_j)-\prod_{j=1}^{i}f(\boldsymbol{h}_j)\prod_{j=i+1}^{t}f(\boldsymbol{g}_j)\right]\right|.$$

By triangle inequality, we have $\Gamma \leq \sum_{i\in[t]} \Gamma_i$. We now proceed with upper bounding Γ_i for a given $i \in [t]$.

$$\begin{aligned}
\Gamma_i &= \left| \mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} \left[\prod_{j=1}^{i-1} f(\boldsymbol{h}_j) \prod_{j=i}^{t} f(\boldsymbol{g}_j) - \prod_{j=1}^{i} f(\boldsymbol{h}_j) \prod_{j=i+1}^{t} f(\boldsymbol{g}_j) \right] \right| \\
&= \left| \mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} \left[(f(\boldsymbol{g}_i) - f(\boldsymbol{h}_i)) \cdot \prod_{j=1}^{i-1} f(\boldsymbol{h}_j) \prod_{j=i+1}^{t} f(\boldsymbol{g}_j) \right] \right| \\
&\leq \sqrt{\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} [(f(\boldsymbol{g}_i) - f(\boldsymbol{h}_i))^2]} \cdot \prod_{j=1}^{i-1} \mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} [f(\boldsymbol{h}_j)^{2(t-1)}]^{\frac{1}{2(t-1)}} \prod_{j=i+1}^{t} \mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} [f(\boldsymbol{g}_j)^{2(t-1)}]^{\frac{1}{2(t-1)}},
\end{aligned}$$

where the last step uses Holder's Inequality. Now, the marginal distribution on each h_j and g_j is identical which is $\mathcal{N}(0,1)^n$, we have

$$\begin{aligned}
\Gamma_i &\leq \sqrt{\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} [(f(\boldsymbol{g}_i) - f(\boldsymbol{h}_i))^2]} \cdot \prod_{j=1}^{i-1} \|f\|_{2(t-1)} \prod_{j=i+1}^{t} \|f\|_{2(t-1)} \\
&\leq \sqrt{\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} [(f(\boldsymbol{g}_i) - f(\boldsymbol{h}_i))^2]} \cdot (\|f\|_{2(t-1)})^{t-1}
\end{aligned}$$

Since a standard one dimensional Gaussian is $(2, q, 1/\sqrt{q-1})$-hypercontractive (Theorem 20), from Proposition 21 , $\|f\|_{2(t-1)} \leq (\sqrt{2t-3})^D \|f\|_2 \leq (\sqrt{2t-3})^D < (2t)^{D/2}$. Thus,

$$\Gamma_i \leq (2t)^{D(t-1)/2} \cdot \sqrt{\mathop{\mathbf{E}}_{\mathcal{H}^{\otimes n}} [(f(\boldsymbol{g}_i) - f(\boldsymbol{h}_i))^2]}$$

Now, each $\boldsymbol{g}_i, \boldsymbol{h}_i$ are such that such that $\mathbf{E}[(\boldsymbol{g}_i)_j \cdot (\boldsymbol{h}_i)_j] = 1 - \delta' \geq 1 - k\delta^2$ for every $j \in [n]$. We can apply Lemma 25 to get $\mathbf{E}_{\mathcal{H}^{\otimes n}}[(f(\boldsymbol{g}_i) - f(\boldsymbol{h}_i))^2] \leq 2k\delta^2 D$. Hence, we can safely upper bound Γ_i as

$$\Gamma_i \leq (2t)^{D(t-1)/2} \cdot 2k\delta D.$$

Therefore, $\Gamma \leq \sum_i \Gamma_i \leq t \cdot (2t)^{D(t-1)/2} \cdot 2k\delta D$ which is at most $2k^2\delta D \cdot (2k)^{Dk/2} \leq \delta \cdot (2k)^{2Dk}$ as required. ◀

B Proof of Observation 8

▶ **Observation 27.** *(Restatement of Observation 8) The distribution $\mathcal{D}_{k,\epsilon}$ above has the following properties:*

1. *$\mathcal{D}_{k,\epsilon}$ is supported on $\mathcal{P}_k$.*
2. *Marginal on every single coordinate is uniform.*
3. *For $i \neq j$, covariance of two variables x_i, x_j sampled form above distribution is:*

$$\mathtt{Cov}[x_i, x_j] = -\frac{\epsilon}{2(1-\alpha)}.$$

4. *If we view $\mathcal{D}_{k,\epsilon}$ as a joint distribution on space $\prod_{i=1}^{k} \mathcal{X}^{(i)}$ where each $\mathcal{X}^{(i)} = \{0,1\}$, then for all $i \in [k]$, $\rho\left(\mathcal{X}^{(i)}, \prod_{j\in[k]\setminus\{i\}} \mathcal{X}^{(j)}; \mathcal{D}_{k,\epsilon}\right) \leq 1 - \frac{\epsilon^2}{2(1-\alpha)^2}$. (See Definition 14 for the definition of ρ.)*

Proof. We prove each of the observations about the distribution. The first property is straight-forward. To prove (2), we compute $\mathbf{E}[x_i]$ as follows.

$$\begin{aligned}\mathbf{E}[x_i] &= (k+1)\cdot\frac{1}{1-\alpha}\left(\frac{1}{k+1}-\epsilon\right)\cdot\frac{1}{2}+\frac{\epsilon}{1-\alpha}\\ &= \frac{1-\epsilon(k+1)+2\epsilon}{2(1-\alpha)}\\ &= \frac{1}{2}\end{aligned}$$

Consider the quantity $\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_i x_j]$. If x is sampled from 0's or e_i's, the value is 0. Moreover, we know that if it is sampled uniformly from $H_k \cup 0^k$, it is $1/4$ because of pairwise independence and the above fact. Therefore, we can write

$$\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_i x_j] = (k+1)\frac{1}{1-\alpha}\left(\frac{1}{k+1}-\epsilon\right)\frac{1}{4}$$

We know that $\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_i] = \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_j] = 1/2$. Therefore,

$$\begin{aligned}\mathtt{Cov}[x_i,x_j] &= \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_i x_j] - \mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_i]\mathop{\mathbf{E}}_{\mathcal{D}_{k,\epsilon}}[x_j]\\ &= \frac{1}{4(1-\alpha)} - \frac{\epsilon(k+1)}{4(1-\alpha)} - \frac{1}{4}\\ &= \frac{-\epsilon}{2(1-\alpha)}\end{aligned}$$

To prove the last item, we first show that the bi-partite graph $G\left(\mathcal{X}^{(i)}, \prod_{j\in[k]\setminus\{i\}}\mathcal{X}^{(j)}, E\right)$ where $(a,b)\in\mathcal{X}^{(i)}\times\prod_{j\in[k]\setminus\{i\}}\mathcal{X}^{(j)}$ is an edge iff $\Pr(a,b)>0$, is connected. To see that the graph is connected, note that for both 0 and 1 on the left hand side, 0^{k-1} is a neighbor on the right hand side as the distribution's support includes e_i for all i, and 0^k. From the distribution, we see that the smallest atom is at least $\frac{\epsilon}{1-\alpha}$, since $\epsilon \le 1/k^2$. We now use Lemma 15 to get the required result. ◀

Forward Analysis for WSTS, Part III: Karp-Miller Trees*

Michael Blondin†[1], Alain Finkel[2], and Jean Goubault-Larrecq[2]

1 **Technische Universität München, Germany**
`blondin@in.tum.de`
2 **LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France**
`finkel@lsv.fr`
3 **LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France**
`goubault@lsv.fr`

Abstract

This paper is a sequel of "Forward Analysis for WSTS, Part I: Completions" [STACS 2009, LZI Intl. Proc. in Informatics 3, 433–444] and "Forward Analysis for WSTS, Part II: Complete WSTS" [Logical Methods in Computer Science 8(3), 2012]. In these two papers, we provided a framework to conduct forward reachability analyses of WSTS, using finite representations of downwards-closed sets. We further develop this framework to obtain a generic Karp-Miller algorithm for the new class of very-WSTS. This allows us to show that coverability sets of very-WSTS can be computed as their finite ideal decompositions. Under natural assumptions on positive sequences, we also show that LTL model checking for very-WSTS is decidable. The termination of our procedure rests on a new notion of acceleration levels, which we study. We characterize those domains that allow for only finitely many accelerations, based on ordinal ranks.

1998 ACM Subject Classification F.1.1. Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases WSTS, model checking, coverability, Karp-Miller algorithm, ideals

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.16

1 Introduction

Context. A well-structured transition system (WSTS) is an infinite well-quasi-ordered set of states equipped with transition relations satisfying one of various possible monotonicity properties. WSTS were introduced in [16] for the purpose of capturing properties common to a wide range of formal models used in verification. Since their inception, much of the work on WSTS has been dedicated to identifying generic classes of WSTS for which verification problems are decidable. Such problems include termination, boundedness [16, 17, 21] and coverability [1, 2, 6, 7]. In general, verifying safety and liveness properties corresponds respectively to deciding the coverability and the repeated control-state reachability problems. Coverability can be decided for WSTS by two different algorithms: the backward algorithm [1, 2] and by combining two forward semi-procedures, one of which enumerates all downwards-closed invariants [25, 6, 7]. Repeated control-state reachability is undecidable for general WSTS, but decidable for Petri nets by use of the Karp-Miller coverability tree [29] and the detection of positive sequences. That technique fails on well-structured extensions of Petri

* A full version of the paper is available at `https://arxiv.org/abs/1710.07258`.
† M. Blondin was supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT).

nets: generating the Karp-Miller tree does not always terminate on ν-Petri nets [36], on reset Petri nets [11], on transfer Petri nets, on broadcast protocols, and on the depth-bounded π-calculus [28, 35, 41] which can simulate reset Petri nets. This is perhaps why little research has been conducted on coverability tree algorithms and model checking of liveness properties for general WSTS. Nonetheless, some recent Petri nets extensions, e.g. ω-Petri nets [24] and unordered data Petri nets [27], benefit from algorithms in the style of Karp and Miller. Hence, there is hope of finding a general framework of WSTS with Karp-Miller-like algorithms.

The Karp-Miller coverability procedure. In 1967, Karp and Miller [29] proposed what is now known as the Karp-Miller coverability tree algorithm, which computes a finite representation (the *clover*) of the downward closure (the *cover*) of the reachability set of a Petri net. In 1978, Valk extended the Karp-Miller algorithm to post-self-modifiying nets [38], a strict extension of Petri nets. In 1987, the second author proposed a generalization of the Karp-Miller algorithm that applies to a class of finitely branching WSTS with strong-strict monotonicity, and having a WSTS completion in which least upper bounds replace the original Petri nets ω-accelerations [16, 17]. In 2004, Finkel, McKenzie and Picaronny [20] applied the framework of [17] to the construction of Karp-Miller trees for strongly increasing ω-recursive nets, a class generalizing post-self-modifiying nets. In 2005, Verma and the third author [40] showed that the construction of Karp-Miller trees can be extended to branching vector addition systems with states. In 2009, the second and the third authors [19] proposed a non-terminating procedure that computes the clover of *any* complete WSTS; this procedure terminates exactly on so-called cover-flattable systems. Recently, this framework has been used for defining computable accelerations in non-terminating Karp-Miller algorithms for both the depth-bounded π-calculus [28] and for ν-Petri nets; terminating Karp-Miller trees are obtained for strict subclasses.

Model checking WSTS. In 1994, Esparza [14] showed that model checking the linear time μ-calculus is decidable for Petri nets by using both the Karp-Miller algorithm and a decidability result due to Valk and Jantzen [39] on infinite T-continual sequences in Petri nets. LTL is undecidable for Petri net extensions such as lossy channel systems [3] and lossy counter machines [37]. In 1998, Emerson and Namjoshi [12] studied the model checking of liveness properties for complete WSTS, but their procedure is not guaranteed to terminate. In 2004, Kouzmin, Shilov and Sokolov [30] gave a generic computability result for a fragment of the μ-calculus; in 2006 and 2013, Bertrand and Schnoebelen [4, 5] studied fixed points in well-structured regular model checking; both [30] and [5] are concerned with formulas with upwards-closed atomic propositions, and do not subsume LTL. In 2011, Chambart, Finkel and Schmitz [9, 10] showed that LTL is decidable for the recursive class of trace-bounded complete WSTS; a class which does not contain all Petri nets.

Our contributions.

- We define *very-well-structured transition systems (very-WSTS)*; a class defined in terms of WSTS completions, and which encompasses models such as Petri nets, ω-Petri nets, post-self-modifying nets and strongly increasing ω-recursive nets. We show that coverability sets of very-WSTS are computable as finite sets of ideals.
- The general clover algorithm of [19], based on the ideal completion studied in [18], does not necessarily terminate and uses an abstract acceleration enumeration. We give an algorithm, the Ideal Karp-Miller algorithm, which organizes accelerations within a tree. We show that this algorithm terminates under natural order-theoretic and effectiveness

conditions, which we make explicit. This allows us to unify various versions of Karp-Miller algorithms in particular particular classes of WSTS.
- We identify the crucial notion of *acceleration level* of an ideal, and relate it to ordinal ranks of sets of reachable states in the completion. We show, notably, that termination is equivalent to the rank being strictly smaller than ω^2. This classifies WSTS into those with high rank (the bad ones), among which those whose sets of states consist of words (e.g., lossy channel systems) or multisets; and those with low rank (the good ones), among which Petri nets and post-self-modifying nets.
- We show that the downward closure of the trace language of a very-WSTS is computable, again as a finite union of ideals. This shows that downward traces inclusion is decidable for very-WSTS.
- Finally, we prove the decidability of model checking liveness properties for very-WSTS under some effectiveness hypotheses.

Differences between very-WSTS and WSTS of [17]. The class of WSTS of [17, Def. 4.17] is reminiscent of very-WSTS. It requires WSTS to be finitely branching and strictly monotone, whereas our definition allows infinite branching and requires the *completion* to be strictly monotone. Moreover, [17, Thm. 4.18], which claims that its Karp-Miller procedure terminates, is incorrect since it does not terminate on transfer Petri nets and broadcast protocols [15], which are finitely branching and strictly monotone WSTS. Finally, some assumptions required to make the Karp-Miller procedure of [17] effective are missing.

Due to space constraints, some proofs are deferred to an extended version of this paper freely available online under the same title.

2 Preliminaries

We write $\subseteq$ for set inclusion and $\subset$ for strict set inclusion. A relation $\leq \subseteq X \times X$ over a set X is a *quasi-ordering* if it is reflexive and transitive, and a *partial ordering* if it is antisymmetric as well. It is *well-founded* if it has no infinite descending chain. A quasi-ordering $\leq$ is a *well-quasi-ordering* (resp. *well partial order*), *wqo* (resp. *wpo*) for short, if for every infinite sequence $x_0, x_1, \cdots \in X$, there exist $i < j$ such that $x_i \leq x_j$. This is strictly stronger than being well-founded.

One example of well-quasi-ordering is the componentwise ordering of tuples over $\mathbb{N}$. More formally, $\mathbb{N}^d$ is well-quasi-ordered by $\leq$ where, for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}^d$, $\boldsymbol{x} \leq \boldsymbol{y}$ if and only if $\boldsymbol{x}(i) \leq \boldsymbol{y}(i)$ for every $i \in [d]$. We extend $\mathbb{N}$ to $\mathbb{N}_\omega \stackrel{\text{def}}{=} \mathbb{N} \cup \{\omega\}$ where $n \leq \omega$ for every $n \in \mathbb{N}_\omega$. $\mathbb{N}_\omega^d$ ordered componentwise is also well-quasi-ordered. Let Σ be a finite alphabet. We denote the set of finite words and infinite words over Σ respectively by Σ^* and Σ^ω. For every $u, v \in \Sigma^*$, we write $u \preceq v$ if u is a subword of v, i.e. u can be obtained from v by removing zero, one or multiple letters. Σ^* is well-quasi-ordered by $\preceq$.

Transition systems. A *(labeled and ordered) transition system* is a triple $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ such that X is a set, Σ is a finite alphabet, $\xrightarrow{a} \subseteq X \times X$ for every $a \in \Sigma$, and $\leq$ is a quasi-ordering on X. Elements of X are called the *states* of $\mathcal{S}$, each $\xrightarrow{a}$ is a *transition relation* of $\mathcal{S}$, and $\leq$ is the *ordering* of $\mathcal{S}$. A *class $\mathcal{C}$ of transition systems* is any set of transition systems. We extend transition relations to sequences over Σ, i.e. for every $x, y \in X$, $x \xrightarrow{\varepsilon} x$, and $x \xrightarrow{wa} y$ if there exists $x' \in X$ such that $x \xrightarrow{w} x' \xrightarrow{a} y$. We write $x \xrightarrow{*} y$ (resp. $x \xrightarrow{+} y$) if there exists $w \in \Sigma^*$ (resp. $w \in \Sigma^+$) such that $x \xrightarrow{w} y$. The finite and infinite *traces* of a transition system $\mathcal{S}$ from

a state $x \in X$ are respectively defined as $\text{Traces}_{\mathcal{S}}(x) \stackrel{\text{def}}{=} \{w \in \Sigma^* : x \xrightarrow{w} y \text{ for some } y \in X\}$ and $\omega\text{-Traces}_{\mathcal{S}}(x) \stackrel{\text{def}}{=} \{w \in \Sigma^\omega : x \xrightarrow{w_1} x_1 \xrightarrow{w_2} \cdots \text{ for some } x_1, x_2, \ldots \in X\}$.

We define the *immediate successors* and *immediate predecessors* of a state x under some sequence $w \in \Sigma^*$ as $\text{Post}_{\mathcal{S}}(x, w) \stackrel{\text{def}}{=} \{y \in X : x \xrightarrow{w} y\}$ and $\text{Pre}_{\mathcal{S}}(x, w) \stackrel{\text{def}}{=} \{y \in X : y \xrightarrow{w} x\}$. The *successors* and *predecessors* of $x \in X$ are $\text{Post}^*_{\mathcal{S}}(x) \stackrel{\text{def}}{=} \{y \in X : x \xrightarrow{*} y\}$ and $\text{Pre}^*_{\mathcal{S}}(x) \stackrel{\text{def}}{=} \{y \in X : y \xrightarrow{*} x\}$. These notations are naturally extended to sets, e.g. $\text{Post}_{\mathcal{S}}(A, w) \stackrel{\text{def}}{=} \{\text{Post}_{\mathcal{S}}(x, w) : x \in A\}$. We say that $\mathcal{S}$ is *deterministic* if $|\text{Post}_{\mathcal{S}}(x, a)| \leq 1$ for every $x \in X$ and $a \in \Sigma$. When $\mathcal{S}$ is deterministic, each $a \in \Sigma$ induces a partial function $t_a : X \to X$ such that $t_a(x) = y$ for each $x \in X$ such that $\text{Post}_{\mathcal{S}}(x, a) = \{y\}$. For readability, we simply write a for t_a, i.e. $a(x) = t_a(x)$. For every $w \in \Sigma^*$, we write $w(x)$ for $\text{Post}_{\mathcal{S}}(x, w)$ if $\text{Post}_{\mathcal{S}}(x, w) \neq \emptyset$.

Well-structured transition systems. A *(labeled and ordered) transition system* $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ is a *well-structured transition system (WSTS)* if $\leq$ is a well-quasi-ordering and $\mathcal{S}$ is *monotone*, i.e. for all $x, x', y \in X$ and $a \in \Sigma$ such that $x \xrightarrow{a} y$ and $x' \geq x$, there exists $y' \in X$ such that $x' \xrightarrow{*} y'$ and $y' \geq y$. Many other types of monotonicities were defined in the literature (see [21]), but, for our purposes, we only need to introduce strong monotonicities. We say that $\mathcal{S}$ has *strong monotonicity* if for all $x, x', y \in X$ and $a \in \Sigma$, $x \xrightarrow{a} y$ and $x' \geq x$ implies $x' \xrightarrow{a} y'$ for some $y' \geq y$. We say that $\mathcal{S}$ has *strong-strict monotonicity*[1] if it has strong monotonicity and for all $x, x', y \in X$ and $a \in \Sigma$, $x \xrightarrow{a} y$ and $x' > x$ implies $x' \xrightarrow{a} y'$ for some $y' > y$.

Verification problems. We say that a *target state* $y \in X$ is *coverable from* an *initial state* $x \in X$ if there exists $z \geq y$ such that $x \xrightarrow{*} z$ and $z \geq y$. The *coverability problem* asks whether a target state y is coverable from an initial state x. The *repeated coverability problem* asks whether a target state y is coverable infinitely often from an initial state x; i.e. whether there exist $z_0, z_1, \cdots \in X$ such that $x \xrightarrow{*} z_0 \xrightarrow{+} z_1 \xrightarrow{+} \cdots$ and $z_i \geq y$ for every $i \in \mathbb{N}$.

3 An investigation of the Karp-Miller algorithm

In order to present our Karp-Miller algorithm for WSTS, we first highlight the key components of the Karp-Miller algorithm for vector addition systems. A *d-dimensional vector addition system (d-VAS)* is a WSTS $\mathcal{V} = (\mathbb{N}^d, \xrightarrow{T}, \leq)$ induced by a finite set $T \subseteq \mathbb{Z}^d$ and the rules:

$$\boldsymbol{x} \xrightarrow{\boldsymbol{t}} \boldsymbol{y} \iff \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{t}, \text{ for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}^d, \boldsymbol{t} \in T.$$

Vector addition systems are deterministic and have strong-strict monotonicity. Given a d-VAS and a vector $\boldsymbol{x}_{\text{init}} \in \mathbb{N}^d$, the Karp-Miller algorithm initializes a rooted tree whose root is labeled by $\boldsymbol{x}_{\text{init}}$. For every $\boldsymbol{t} \in T$ such that $\boldsymbol{x} + \boldsymbol{t} \geq \boldsymbol{0}$, a child labeled by $\boldsymbol{x} + \boldsymbol{t}$ is added to the root. This process is repeated successively to the new nodes. If a newly added node $c : \boldsymbol{x}$ has an ancestor $c' : \boldsymbol{x}'$ such $\boldsymbol{x} = \boldsymbol{x}'$, then it is not explored furthermore. If a newly added node $c : \boldsymbol{x}$ has an ancestor $c' : \boldsymbol{x}'$ such $\boldsymbol{x} > \boldsymbol{x}'$, then c is relabeled by the vector $\boldsymbol{y} \in \mathbb{N}^d_\omega$ such that $\boldsymbol{y}(i) \stackrel{\text{def}}{=} \boldsymbol{x}(i)$ if $\boldsymbol{x}(i) = \boldsymbol{x}'(i)$ and $\boldsymbol{y}(i) \stackrel{\text{def}}{=} \omega$ if $\boldsymbol{x}(i) > \boldsymbol{x}'(i)$. The latter operation is called an *acceleration* of c.

[1] Strong-strict monotonicity should not be confused with strong *and* strict monotonicities. Here strongness and strictness have to hold at the *same* time.

A vector $\boldsymbol{x}_{\text{tgt}}$ is coverable from $\boldsymbol{x}_{\text{init}}$ if and only if the resulting tree $\mathcal{T}$ contains a node $c : \boldsymbol{x}$ such that $\boldsymbol{x} \geq \boldsymbol{x}_{\text{tgt}}$. Similarly, $\boldsymbol{x}_{\text{tgt}}$ is repeatedly coverable from $\boldsymbol{x}_{\text{init}}$ if and only if $\mathcal{T}$ contains a node $c : \boldsymbol{x}$ that has an ancestor that was accelerated, and such that $\boldsymbol{x} \geq \boldsymbol{x}_{\text{tgt}}$.

3.1 Ideals and completions

One feature of the Karp-Miller algorithm is that it works over $\mathbb{N}_\omega^d$ instead of $\mathbb{N}^d$. Intuitively, vectors containing some ω correspond to "limit" elements. For a generic WSTS $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$, a similar extension of X is not obvious. Let us present one, called the *completion* of $\mathcal{S}$ in [19]. Instead of operating over X, the completion of $\mathcal{S}$ operates over the so-called *ideals* of X. In particular, the ideals of $\mathbb{N}^d$ are isomorphic to $\mathbb{N}_\omega^d$.

Let X be a set quasi-ordered by $\leq$. The *downward closure* of $D \subseteq X$ is defined as $\downarrow D \stackrel{\text{def}}{=} \{x \in X : x \leq y \text{ for some } y \in D\}$. A subset $D \subseteq X$ is *downwards-closed* if $D = \downarrow D$. An *ideal* is a downwards-closed subset $I \subseteq X$ that is additionally *directed*: I is non-empty and for all $x, y \in I$, there exists $z \in I$ such that $x \leq z$ and $y \leq z$ (equivalently, every finite subset of I has an upper bound in I). We denote the set of ideals of X by $\text{Idl}(X)$, i.e. $\text{Idl}(X) \stackrel{\text{def}}{=} \{D \subseteq X : D = \downarrow D \text{ and } D \text{ is directed}\}$.

It is known that $\text{Idl}(\mathbb{N}^d) = \{A_1 \times \cdots \times A_d : A_1, \ldots, A_d \in \{\downarrow n : n \in \mathbb{N}\} \cup \{\mathbb{N}\}\}$. Therefore, every ideal of $\mathbb{N}^d$ is naturally represented by some vector of $\mathbb{N}_\omega^d$, and vice versa. We write $\omega\text{-rep}(I)$ for this representation, for every $I \in \text{Idl}(\mathbb{N}^d)$. For example, the ideal $I = \mathbb{N} \times \downarrow 8 \times \downarrow 3 \times \mathbb{N}$ is represented by $\omega\text{-rep}(I) = (\omega, 8, 3, \omega)$.

Downwards-closed subsets can often be represented by finitely many ideals:

▶ **Theorem 1** ([13, 8, 33, 34, 23, 31]). *Let X be a well-quasi-ordered set. For every downwards-closed subset $D \subseteq X$, there exist $I_1, I_2, \ldots, I_n \in \text{Idl}(X)$ s.t. $D = I_1 \cup I_2 \cup \cdots \cup I_n$.*

This theorem gives rise to a canonical decomposition of downwards-closed sets. The *ideal decomposition* of a downwards-closed subset $D \subseteq X$ is the set of maximal ideals contained in D with respect to inclusion. We denote the ideal decomposition of D by $\text{IdealDecomp}(D) \stackrel{\text{def}}{=} \max_{\subseteq}\{I \in \text{Idl}(X) : I \subseteq D\}$. By Theorem 1, $\text{IdealDecomp}(D)$ is finite, and $D = \bigcup_{I \in \text{IdealDecomp}(D)} I$. In [19, 6], the notion of ideal decomposition is used to define the completion of unlabeled WSTS. We slightly extend this notion to labeled WSTS:

▶ **Definition 2.** Let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a labeled WSTS. The *completion* of $\mathcal{S}$ is the labeled transition system $\widehat{\mathcal{S}} = (\text{Idl}(X), \overset{\Sigma}{\rightsquigarrow}, \subseteq)$ such that $I \overset{a}{\rightsquigarrow} J$ if, and only if,

$$J \in \text{IdealDecomp}(\downarrow \text{Post}_{\mathcal{S}}(I, a)).$$

The completion of a WSTS enjoys numerous properties. In particular, it has strong monotonicity, and it is finitely branching [6], i.e. $\text{Post}_{\widehat{\mathcal{S}}}(I, a)$ is finite for every $I \in \text{Idl}(X)$ and $a \in \Sigma$. Note that if $\mathcal{S}$ has strong-strict monotonicity, then this property is not necessarily preserved by $\widehat{\mathcal{S}}$ [6]. Moreover, the completion of a WSTS may not be a WSTS since $\text{Idl}(X)$ is not always well-quasi-ordered by $\subseteq$. However, for the vast majority of models used in verification, $\text{Idl}(X)$ is well-quasi-ordered, and hence completions remain well-structured. Indeed, $\text{Idl}(X)$ is well-quasi-ordered if and only if X is a so-called ω^2-wqo, and all known wqos, except possibly graphs under minor embedding, are ω^2-wqo, as discussed in [19]. The traces of a WSTS are closely related to those of its completion:

▶ **Proposition 3** ([6]). *The following holds for every WSTS $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$:*

1. *For all $x, y \in X$ and $w \in \Sigma^*$, if $x \xrightarrow{w} y$, then for every ideal $I \supseteq \downarrow x$, there exists an ideal $J \supseteq \downarrow y$ such that $I \overset{w}{\rightsquigarrow} J$.*

2. *For all* $I, J \in \mathrm{Idl}(X)$ *and* $w \in \Sigma^*$*, if* $I \overset{w}{\rightsquigarrow} J$*, then for every* $y \in J$*, there exist* $x \in I, y' \in X$ *and* $w' \in \Sigma^*$ *such that* $x \xrightarrow{w'} y'$ *and* $y' \geq y$*. If* $\mathcal{S}$ *has strong monotonicity, then* $w' = w$*.*
3. *if* $\mathcal{S}$ *has strong monotonicity, then* $\bigcup_{J \in Post_{\widehat{\mathcal{S}}}(I,w)} J = {\downarrow} Post_{\mathcal{S}}(I, w)$ *for all* $I \in \mathrm{Idl}(X)$ *and* $w \in \Sigma^*$*.*
4. *if* $\mathcal{S}$ *has strong monotonicity, then* $\mathrm{Traces}_{\mathcal{S}}(x) = \mathrm{Traces}_{\widehat{\mathcal{S}}}({\downarrow} x)$ *and* $\omega\text{-}\mathrm{Traces}_{\mathcal{S}}(x) \subseteq \omega\text{-}\mathrm{Traces}_{\widehat{\mathcal{S}}}({\downarrow} x)$ *for every* $x \in X$*.*

It is worth noting that if $\mathcal{S}$ is infinitely branching, then an infinite trace of $\widehat{\mathcal{S}}$ from ${\downarrow} x$ is not necessarily an infinite trace of $\mathcal{S}$ from x (e.g. see [6]). Whenever the completion of a WSTS $\mathcal{S}$ is deterministic, we will often write $w(I)$ for $\mathrm{Post}_{\widehat{\mathcal{S}}}(I, w)$ if the latter is nonempty and if there is no ambiguity with $\mathrm{Post}_{\mathcal{S}}(I, w)$.

3.2 Levels of ideals

The Karp-Miller algorithm terminates for the following reasons: $\mathbb{N}^d_\omega$ is well-quasi-ordered and ω's can only be added to vectors along a branch at most d times. Loosely speaking, the latter property means that $\mathrm{Idl}(\mathbb{N}^d)$ has $d+1$ "levels". Here, we generalize this notion. We say that an infinite sequence of ideals $I_0, I_1, \ldots \in \mathrm{Idl}(X)$ is an *acceleration candidate* if $I_0 \subset I_1 \subset \cdots$.

▶ **Definition 4.** For every $n \in \mathbb{N}$, the n^{th} *level* of $\mathrm{Idl}(X)$ is defined as

$$\mathrm{Acc}_n(X) = \begin{cases} \mathrm{Idl}(X) & \text{if } n = 0, \\ \left\{ \bigcup_{i \in \mathbb{N}} I_i : I_0, I_1, \ldots \in \mathrm{Acc}_{n-1}(X) \text{ is an acceleration candidate} \right\} & \text{if } n > 0. \end{cases}$$

We observe that $\mathrm{Acc}_{n+1}(X) \subseteq \mathrm{Acc}_n(X)$ for every $n \in \mathbb{N}$. Moreover, as expected:

$$\mathrm{Acc}_n(\mathbb{N}^d) = \{I \in \mathrm{Idl}(\mathbb{N}^d) : \omega\text{-}rep(I) \text{ has at least } n \text{ occurrences of } \omega\}.$$

We say that $\mathrm{Idl}(X)$ has *finitely many levels* if there exists $n \in \mathbb{N}$ such that $\mathrm{Acc}_n(X) = \emptyset$. For example, $\mathrm{Acc}_{d+1}(\mathbb{N}^d) = \emptyset$.

3.3 Accelerations

The last key aspect of the Karp-Miller algorithm is the possibility to accelerate nodes. In order to generalize this notion, let us briefly develop some intuition. Recall that a newly added node $c : \boldsymbol{x}$ is accelerated if it has an ancestor $c' : \boldsymbol{x}'$ such that $\boldsymbol{x} > \boldsymbol{x}'$. Consider the non-empty sequence w labeling the path from c' to c. Since d-VAS have strong-strict monotonicity, both over $\mathbb{N}^d$ and $\mathbb{N}^d_\omega$, $w^n(\boldsymbol{x})$ is defined for every $n \in \mathbb{N}$. For example, if $(5,0,1) \xrightarrow{w} (5,1,3)$ is encountered, $(5,1,3)$ is replaced by $(5,\omega,\omega)$. This represents the fact that for every $n \in \mathbb{N}$, there exists some reachable marking $\boldsymbol{y} \geq (5,n,n)$. Note that an acceleration increases the number of occurrences of ω. In our example, the ideal $I = {\downarrow}5 \times {\downarrow}1 \times {\downarrow}3$, which is of level 0, is replaced by $I' = {\downarrow}5 \times \mathbb{N} \times \mathbb{N}$, which is of level 2. Based on these observations, we extend the notion of acceleration to completions:

▶ **Definition 5.** Let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a WSTS such that $\widehat{\mathcal{S}}$ is deterministic and has strong-strict monotonicity, let $w \in \Sigma^+$ and let $I \in \mathrm{Idl}(X)$. The *acceleration* of I under w is defined as:

$$w^\infty(I) \overset{\text{def}}{=} \begin{cases} \bigcup_{k \in \mathbb{N}} w^k(I) & \text{if } I \subset w(I), \\ I & \text{otherwise.} \end{cases}$$

Note that for every ideal I, $w^\infty(I)$ is also an ideal. As for $\mathrm{Idl}(\mathbb{N}^d)$, any successor J of an ideal I belongs to the same level of I, and accelerating an ideal increases its level.

▶ **Proposition 6.** *Let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a WSTS such that $\mathcal{S}$ has strong monotonicity, and $\widehat{\mathcal{S}}$ is deterministic and has strong-strict monotonicity. For every $I \in \mathrm{Idl}(X)$ and $w \in \Sigma^+$,*
1. *if $\mathrm{Post}_{\widehat{\mathcal{S}}}(I, w) \neq \emptyset$ and $I \in \mathrm{Acc}_n(X)$ for some $n \in \mathbb{N}$, then $w(I) \in \mathrm{Acc}_n(X)$;*
2. *if $I \subset w(I)$ and $I \in \mathrm{Acc}_n(X)$ for some $n \in \mathbb{N}$, then $w^\infty(I) \in \mathrm{Acc}_{n+1}(X)$.*

4 The Ideal Karp-Miller algorithm

We may now present our generalization of the Karp-Miller algorithm. To do so, we first define the class of WSTS that enjoys all of the properties introduced in the previous section:

▶ **Definition 7.** A *very-WSTS* is a labeled WSTS $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ such that:
- $\mathcal{S}$ has strong monotonicity,
- $\widehat{\mathcal{S}}$ is a deterministic WSTS with strong-strict monotonicity,
- $\mathrm{Idl}(X)$ has finitely many levels.

The class of very-WSTS includes vector addition systems, Petri nets, ω-Petri nets [24], post-self-modifying nets [38] and strongly increasing ω-recursive nets [20]. However, very-WSTS do not include transfer Petri nets, since $\widehat{S}$ does not have strict monotonicity, and unordered data Petri nets, since $\mathrm{Idl}(X)$ has infinitely many levels. Note that $\widehat{\mathcal{S}}$ may be deterministic (and finitely branching) even when $\mathcal{S}$ is not, and even when $\mathcal{S}$ is not finitely branching, as the example of ω-Petri nets shows.

We present the *Ideal Karp-Miller algorithm (IKM)* for this class in Algorithm 4.1. The algorithm starts from an ideal I_0, successively computes its successors in $\widehat{\mathcal{S}}$ and performs accelerations as in the classical Karp-Miller algorithm for VAS. Note that we do *not* allow for nested accelerations. For every node $c : \langle I, n\rangle$ of the tree built by the algorithm, we write ideal(c) for I, and num-accel(c) for n, which will be the number of accelerations made along the branch from the root to c (inclusively). Let us first show that the algorithm terminates.

Algorithm 4.1: Ideal Karp-Miller algorithm.

```
1  initialize a tree T with root r: ⟨I_0, 0⟩
2  while T contains an unmarked node c: ⟨I, n⟩ do
3      if c has an ancestor c': ⟨I', n'⟩ s.t. I' = I then mark c
4      else
5          if c has an ancestor c': ⟨I', n'⟩ s.t. I' ⊂ I
6                and n' = n /* no acceleration occurred between c' and c */
           then
7              w ← sequence of labels from c' to c
8              replace c: ⟨I, n⟩ by c: ⟨w^∞(I), n + 1⟩
9          for a ∈ Σ do
10             if a(I) is defined then
11                 add arc labeled by a from c to a new child d: ⟨a(I), n⟩
12         mark c
13 return T
```

▶ **Theorem 8.** *Algorithm 4.1 terminates for very-WSTS.*

Proof. We note the following invariants: (1) for every node $c : \langle I, n\rangle$ of $\mathcal{T}$, I is in $\mathrm{Acc}_n(X)$; (2) at line 2, i.e., each time control returns to the beginning of the loop, all unmarked nodes of $\mathcal{T}$ are leaves; (3) num-accel(c) is non-decreasing on each branch of $\mathcal{T}$, that is: for every branch $c_0 : \langle I_0, n_0\rangle$, $c_1 : \langle I_1, n_1\rangle$, …, $c_k : \langle I_k, n_k\rangle$ of $\mathcal{T}$, we have $n_1 \leq n_2 \leq \cdots \leq n_k$. (1) is by Proposition 6, (2) is an easy induction on the number of times through the loop, and (3) is also by induction, noticing that by (2) only n_k can increase when line 8 is executed.

The rest of the argument is as for the classical Karp-Miller algorithm. Suppose the algorithm does not terminate. Let $\mathcal{T}_n$ be the finite tree obtained after n iterations. The infinite sequence $\mathcal{T}_0, \mathcal{T}_1, \ldots$ defines a unique infinite tree $\mathcal{T}_\infty = \bigcup_{n\in\mathbb{N}} \mathcal{T}_n$. Since $\widehat{\mathcal{S}}$ is finitely branching, $\mathcal{T}_\infty$ is also finitely branching. Therefore, $\mathcal{T}_\infty$ contains an infinite path $c_0 : \langle I_0, n_0\rangle$, $c_1 : \langle I_1, n_1\rangle$, …, $c_k : \langle I_k, n_k\rangle$, …, by König's lemma. By (1), and since $\mathrm{Idl}(X)$ has finitely many levels, the numbers n_k assume only finitely many values. Let N be the largest of those values. Using (3), there is a $k_0 \in \mathbb{N}$ such that $n_k = N$ for every $k \geq k_0$. Since $\widehat{\mathcal{S}}$ is a WSTS, hence $\mathrm{Idl}(X)$ is wqo, we can find two indices i, j with $k_0 \leq i < j$ and such that $I_i \subseteq I_j$. If $I_i = I_j$, then line 3 of the algorithm would have stopped the exploration of the path. Hence $I_i \subset I_j$, but then line 8 would have replaced num-accel$(c_j) = N$ by $N + 1$, contradiction. ◀

4.1 Properties of the algorithm

Let $\mathcal{T}_I$ denote the tree induced by the set of nodes returned by Algorithm 4.1 on input $(\mathcal{S}, I)$. Let $D_I \stackrel{\text{def}}{=} \bigcup_{c\in\mathcal{T}_I} \mathrm{ideal}(c)$. We claim that $D_I = {\downarrow}\mathrm{Post}^*_{\mathcal{S}}(I)$. Instead of proving this claim directly, we take traces into consideration and prove a stronger statement. We define two word automata that will be useful for this purpose.

▶ **Definition 9.** The *stuttering automaton*[2] is the finite word automaton $\mathcal{A}_I$ obtained by making all of the states of $\mathcal{T}_I$ accepting, by taking the root r as the initial state, and by taking the arcs of $\mathcal{T}_I$ as transitions, together with the following additional transitions:

- If a leaf c of $\mathcal{T}_I$ has an ancestor c' such that $\mathrm{ideal}(c) = \mathrm{ideal}(c')$, then a transition from c to c' labeled by ε is added to $\mathcal{A}_I$.

The *Karp-Miller automaton* is the automaton $\mathcal{K}_I$ obtained by extending $\mathcal{A}_I$ as follows:

- If a node c of $\mathcal{T}_I$ has been accelerated because of an ancestor c', then a transition from c to c' labeled by ε is added to $\mathcal{K}_I$.

Both $\mathcal{A}_I$ and $\mathcal{K}_I$ can be computed from $\mathcal{T}_I$. Moreover, they give precious information about the traces of $\mathcal{S}$. Let $\mathrm{L}(\mathcal{A}_I)$ and $\mathrm{L}(\mathcal{K}_I)$ denote the language over Σ accepted by $\mathcal{A}_I$ and $\mathcal{K}_I$. We will show the following theorem:

▶ **Theorem 10.** *For every very-WSTS $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ and $I \in \mathrm{Idl}(X)$,*

$$D_I = {\downarrow}\mathit{Post}^*_{\mathcal{S}}(I),\ \mathrm{Traces}_{\mathcal{S}}(I) \subseteq \mathrm{L}(\mathcal{A}_I) \text{ and } \mathrm{L}(\mathcal{K}_I) \subseteq {\downarrow_\preceq}\,\mathrm{Traces}_{\mathcal{S}}(I).$$

In particular, for every $x \in X$, $D_{\downarrow x} = {\downarrow}\mathit{Post}^*_{\mathcal{S}}(x)$, ${\downarrow_\preceq}\,\mathrm{L}(\mathcal{K}_{\downarrow x}) = {\downarrow_\preceq}\,\mathrm{Traces}_{\mathcal{S}}(x)$, and ${\downarrow_\preceq}\,\mathrm{Traces}_{\mathcal{S}}(x)$ is a computable regular language.

The proof of Theorem 10 follows from the forthcoming Prop. 11 describing the relations between traces of $\mathcal{A}_I$ and $\mathcal{K}_I$ with traces of $\mathcal{S}$ and $\widehat{\mathcal{S}}$. We write $c \stackrel{w}{\dashrightarrow}_{\mathcal{T}} c'$, $c \stackrel{w}{\dashrightarrow}_{\mathcal{A}} c'$ and $c \stackrel{w}{\dashrightarrow}_{\mathcal{K}} c'$ whenever node c' can be reached by reading w from c in $\mathcal{T}_I$, $\mathcal{A}_I$ and $\mathcal{K}_I$ respectively.

[2] We use the term *stuttering* as paths of the automaton correspond to stuttering paths of [24].

▶ **Proposition 11.** *Let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a very-WSTS and let $I_0 \in \mathrm{Idl}(X)$.*

1. *For every $y, z \in X$, $w \in \Sigma^*$ and $c \in \mathcal{A}_{I_0}$, if $y \xrightarrow{w} z$ and $y \in \mathrm{ideal}(c)$, then there exists $d \in \mathcal{A}_{I_0}$ such that $c \overset{w}{\dashrightarrow}_{\mathcal{A}} d$ and $z \in \mathrm{ideal}(d)$.*
2. *For every $z \in X$, $w \in \Sigma^*$ and $c, d \in \mathcal{K}_{I_0}$, if $c \overset{w}{\dashrightarrow}_{\mathcal{K}} d$ and $z \in \mathrm{ideal}(d)$, then there exist $y \in \mathrm{ideal}(c)$, $w' \succeq w$ and $z' \geq z$ such that $y \xrightarrow{w'} z'$.*

Proof. We only prove (2). The proof is by induction on $|w|$. If $|w| = 0$, then $w = \varepsilon$. We stress the fact that even though w is empty, d might differ from c since $\mathcal{K}_{I_0}$ contains ε-transitions. However, by definition of $\mathcal{K}_{I_0}$, we know that $\mathrm{ideal}(d) \subseteq \mathrm{ideal}(c)$. Therefore, $z \in \mathrm{ideal}(c)$, and we are done since $z \xrightarrow{\varepsilon} z$.

Suppose that $|w| > 0$. Assume the claim holds for every word of length less than $|w|$. There exist $u, v \in \Sigma^*$, $a \in \Sigma$ and $d' \in \mathcal{K}_{I_0}$ such that $w = uav$, $c \overset{u}{\dashrightarrow}_{\mathcal{K}} d' \overset{a}{\dashrightarrow}_{\mathcal{K}} d \overset{v}{\dashrightarrow}_{\mathcal{K}} d$ and d' is the parent of d in T_{I_0}. Let $I \overset{\text{def}}{=} \mathrm{ideal}(c)$, $J \overset{\text{def}}{=} \mathrm{ideal}(d')$, $K \overset{\text{def}}{=} \mathrm{ideal}(d)$, and $K' \overset{\text{def}}{=} a(J)$. By induction hypothesis, there exist $y_K \in K$, $v' \succeq v$ and $z' \geq z$ such that $y_K \xrightarrow{v'} z'$.

- If $K = K'$, then $J \overset{a}{\rightsquigarrow} K$. By definition of $\overset{a}{\rightsquigarrow}$, there exist $y_J \in J$ and $y'_K \geq y_K$ such that $y_J \xrightarrow{a} y'_K$. By induction hypothesis, there exist $y_I \in I$, $u' \succeq u$ and $y'_J \geq y_J$ such that $y_I \xrightarrow{u'} y'_J$. By strong monotonicity of $\mathcal{S}$, there exists $z'' \geq z'$ such that $y_I \xrightarrow{u'av'} z''$. We are done since $u'av' \succeq uav$.
- If $K \neq K'$, then K was obtained through an acceleration. Therefore, $K = \sigma^\infty(K')$ for some $\sigma \in \Sigma^+$. This implies that $y_K \in \sigma^k(K')$ for some $k \in \mathbb{N}$. Let $L \overset{\text{def}}{=} \sigma^k(K')$. Note that $J \overset{a}{\rightsquigarrow} K' \overset{\sigma^k}{\rightsquigarrow} L$. By Prop. 3(2), there exist $y_J \in J$ and $y'_K \geq y_K$ such that $y_J \xrightarrow{a\sigma^k} y'_K$. By induction hypothesis, there exist $y_I \in I$, $u' \succeq u$ and $y'_J \geq y_J$ such that $y_I \xrightarrow{u'} y'_J$. By strong monotonicity of $\mathcal{S}$, there exists $z'' \geq z'$ such that $y_I \xrightarrow{u'av^kv'} z''$. ◀

4.2 Effectiveness of the algorithm

The Ideal Karp-Miller algorithm can be implemented provided that (1) ideals can be effectively manipulated, (2) inclusion of ideals can be tested, (3) $\mathrm{Post}_{\mathcal{S}}(I)$ can be computed for every ideal I, and (4) $w^\infty(I)$ can be computed for every ideal I and sequence w. A class of WSTS satisfying (1–3) is called *completion-post-effective*, and a class satisfying (4) is called *∞-completion-effective*. By Theorem 10, we obtain the following result:

▶ **Theorem 12.** *Let $\mathcal{C}$ be a completion-post-effective and ∞-completion-effective class of very-WSTS. The ideal decomposition of $\downarrow \mathrm{Post}^*_{\mathcal{S}}(x)$ can be computed for every $\mathcal{S} = (X, \to, \leq) \in C$ and $x \in X$. In particular, coverability for $\mathcal{C}$ is decidable.*

5 A characterization of acceleration levels

We pause for a moment, and give a precise characterization of ideals that have finitely many levels. We shall then discuss some extensions briefly, beyond the finitely many level case.

Let Z be a well-founded partially ordered set, abstracting away from the case $Z = \mathrm{Idl}(X)$. The *rank* of $z \in Z$, denoted $\mathrm{rk}\, z$, is the ordinal defined inductively by $\mathrm{rk}\, z \overset{\text{def}}{=} \sup\{\mathrm{rk}\, y + 1 : y < z\}$, where $\sup(\emptyset) \overset{\text{def}}{=} 0$. The *rank* of Z is defined as $\mathrm{rk}\, Z \overset{\text{def}}{=} \sup\{\mathrm{rk}\, z + 1 : z \in Z\}$. We say that a sequence $z_0, z_1, \ldots \in Z$ is an *acceleration candidate* if $z_1 < z_2 < \cdots < z_i < \cdots$. Such an acceleration candidate *goes through* a set A if $z_i \in A$ for some $i \in \mathbb{N}$, and is *below* $z \in Z$ if $z_i \leq z$ for every $i \in \mathbb{N}$. We define a family of sets $A_\alpha(Z)$ closely related to levels of ideals:

▶ **Definition 13.** Let Z be a partially ordered set. Let $A_0(Z) \stackrel{\text{def}}{=} \emptyset$. For every ordinal $\alpha > 0$, $A_\alpha(Z)$ is the set of elements $z \in Z$ such that every acceleration candidate below z goes through $A_\beta(Z)$ for some $\beta < \alpha$.

Observe that $A_\alpha(Z) \subseteq A_\beta(Z)$ for every $\alpha \leq \beta$, and that $A_n(\mathbb{N}_\omega^d)$ is the set of d-tuples with less than n components equal to ω. It is easily shown that $A_n(\mathrm{Idl}(X))$ is the upward closure of the complement of $\mathrm{Acc}_n(X)$. Consequently, $A_n(\mathrm{Idl}(X)) = \mathrm{Idl}(X)$ if and only if $\mathrm{Acc}_n(X) = \emptyset$, and we can bound levels of $\mathrm{Idl}(X)$ by means of $A_n(\mathrm{Idl}(X))$.

Let us first show that $A_n(Z)$ is exactly the set of elements of rank less than $\omega \cdot n$. This rests on the following, which is perhaps less obvious than it seems.

▶ **Lemma 14.** *Let Z be a countable wpo. For every $z \in Z$ such that $\mathrm{rk}\, z$ is a limit ordinal, z is the supremum of some acceleration candidate $z_0 < z_1 < \cdots$. Moreover, for any given ordinal $\beta < \mathrm{rk}\, z$, the acceleration candidate can be chosen such that $\beta \leq z_i$ for every $i \in \mathbb{N}$.*

This fails if Z is not countable: take $Z = \omega_1 + 1$, where ω_1 is the first uncountable ordinal, then $\omega_1 \in Z$ is not the supremum of countably many ordinals $< \omega_1$. This also fails if Z is not wqo, even when Z is well-founded: consider the set with one root r above chains of length n, one for each $n \in \mathbb{N}$: $\mathrm{rk}\, r = \omega$, but there is no acceleration candidate below r.

Proof. Let $\alpha \stackrel{\text{def}}{=} \mathrm{rk}\, z$. A *fundamental sequence* for α is a monotone sequence of ordinals strictly below α whose supremum equals α. Fundamental sequences exist for all countable limit ordinals, in particular for α, since Z is countable (e.g. see [22]). Pick one such fundamental subsequence $(\gamma_i)_{i\in\mathbb{N}}$. Replacing γ_i by $\sup(\beta, \gamma_i)$ if necessary, we may assume that $\beta \leq \gamma_m$ for every $i \in \mathbb{N}$. By the definition of rank, for every $i \in \mathbb{N}$, there is an element $z_i < z$ of rank at least γ_i. Since Z is well-quasi-ordered, we may extract a non-decreasing subsequence from $(z_i)_{i\in\mathbb{N}}$. Without loss of generality, assume that $z_0 \leq z_1 \leq \cdots$. If all but finitely many of these inequalities were equalities, then z would be equal to z_i for m large enough, but that is impossible since $z_i < z$. We can therefore extract a strictly increasing subsequence from $(z_i)_{i\in\mathbb{N}}$. This is an acceleration sequence, its supremum is z, and $\beta \leq \gamma_i \leq z_i$ for every i. ◀

▶ **Lemma 15.** *Let Z be a countable wpo, and let $n \in \mathbb{N}$. For every $z \in Z$, $\mathrm{rk}\, z < \omega \cdot n$ if and only if $z \in A_n(Z)$.*

Proof. $\Rightarrow$) By induction on n. The case $n = 0$ is immediate. Let $n \geq 1$. Given any acceleration candidate $z_1 < z_2 < \cdots$ below z, we must have $\mathrm{rk}\, z_1 < \mathrm{rk}\, z_2 < \cdots < \mathrm{rk}\, z$. Since $\mathrm{rk}\, z < \omega \cdot n$, there exist $\ell, m \in \mathbb{N}$ with $\ell < n$ such that $\mathrm{rk}\, z = \omega.\ell + m$. Therefore, $\mathrm{rk}\, z_i \geq \omega \cdot \ell$ for only finitely many i. In particular, there exists some i such that $\mathrm{rk}\, z_i < \omega \cdot \ell$. Since $\ell < n$, we have $\mathrm{rk}\, z_i < \omega \cdot (n-1)$. By induction hypothesis, $z_i \in A_{n-1}(Z)$, and hence $z \in A_n(Z)$.

$\Leftarrow$) We show by induction on n that $\mathrm{rk}\, z \geq \omega \cdot n$ implies $z \notin A_n(Z)$. The case $n = 0$ is immediate. Let $n \geq 1$. In general, $\mathrm{rk}\, z$ is not a limit ordinal, but can be written as $\alpha + \ell$ for some limit ordinal α and some $\ell \in \mathbb{N}$. By definition of rank, z is larger than some element of rank $\alpha + (\ell - 1)$, which is itself larger than some element of rank $\alpha + (\ell - 2)$, and so on. Iterating this way, we find an element $y \leq z$ of rank exactly α. Since $\mathrm{rk}\, y$ is a limit ordinal, Lemma 14 entails that y is the supremum of some acceleration candidate $z_0 < z_1 < \cdots$. Moreover, since $\omega \cdot (n-1) < \mathrm{rk}\, y$, we may assume that $\mathrm{rk}\, z_i \geq \omega \cdot (n-1)$ for every $i \in \mathbb{N}$. By induction hypothesis, $z_i \notin A_{n-1}(Z)$ for every $i \in \mathbb{N}$, and hence $z \notin A_n(Z)$. ◀

▶ **Theorem 16.** *Let X be a countable wqo such that* $\mathrm{Idl}(X)$ *is well-quasi-ordered by inclusion*[3]. *The following holds:* $\mathrm{Idl}(X)$ *has finitely many levels if and only if* $\mathrm{rk}\,\mathrm{Idl}(X) < \omega^2$.

Proof. We apply Lemma 15 to $Z = \mathrm{Idl}(X)$, a wpo by assumption. For that, we need to show that Z is countable. There are countably many upwards-closed subsets, since they are all determined by their finitely many minimal elements. Downwards-closed subsets are in one-to-one correspondence with upwards-closed subsets, through complementation, hence are countably many as well, and ideals are particular downwards-closed subsets.

We conclude by noting that the following are equivalent: (1) $\mathrm{rk}\,\mathrm{Idl}(X) < \omega^2$; (2) $\mathrm{rk}\,\mathrm{Idl}(X) \leq \omega \cdot n$ for some $n \in \mathbb{N}$; (3) $A_n(\mathrm{Idl}(X)) = \mathrm{Idl}(X)$ for some $n \in \mathbb{N}$ (by Lemma 15); (4) $\mathrm{Acc}_n(X) = \emptyset$ for some $n \in \mathbb{N}$. ◀

While $\mathrm{rk}\,\mathrm{Idl}(\mathbb{N}^d) = \omega \cdot d + 1 < \omega^2$, not all wqos X used in verification satisfy $\mathrm{rk}\,\mathrm{Idl}(X) < \omega^2$. For example, $\mathrm{rk}\,\mathrm{Idl}(\Sigma^*) = \omega^{|\Sigma|}+1$, for any finite alphabet Σ; a similar result holds for multisets over Σ.

Note that the IKM algorithm still terminates if, for each branch $B = (c_0 : \langle I_0, n_0\rangle, c_1 : \langle I_1, n_1\rangle, \ldots, c_k : \langle I_k, n_k\rangle, \ldots)$ of the Ideal Karp-Miller tree, $[B] \stackrel{\text{def}}{=} \{I \in \mathrm{Idl}(X) : \exists j, k \in \mathbb{N}, j \leq k \text{ and } I_j \subseteq I \subseteq I_k\}$ has rank less than ω^2. Indeed, the IKM algorithm terminates if and only if each branch B is finite, and the states involved in computing the branch, as well as all needed accelerations, are all included in $[B]$. Therefore, relaxing "$\mathrm{rk}\,\mathrm{Idl}(X) < \omega^2$" to the more technical condition "$\mathrm{rk}\,[B] < \omega^2$" may allow one to extend the notion of very-WSTS.

6 Model checking liveness properties for very-WSTS

In this section, we show how the Ideal Karp-Miller algorithm can be used to test whether a very-WSTS violates a liveness property specified by an LTL formula. Testing that $\mathcal{S}$ violates a property φ amounts to constructing a Büchi automaton $\mathcal{B}_{\neg\varphi}$ for $\neg\varphi$ and to test whether $\mathcal{B}_{\neg\varphi}$ accepts an infinite trace of $\mathcal{S}$. We first introduce positive very-WSTS, and show that repeated coverability is decidable for them under some effectiveness hypothesis. Then, we show how LTL model checking for positive very-WSTS reduces to repeated coverability.

6.1 Deciding repeated coverability

Let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a WSTS and let $x \in X$. We say that $w \in \Sigma^*$ is *positive for* x if there exists some $y \in X$ such that $x \xrightarrow{w} y$ and $x \leq y$. We say that $w \in \Sigma^*$ is *positive* if w is positive for every $x \in X$ such that $\mathrm{Post}\,(x, w) \neq \emptyset$. We say that a WSTS $\mathcal{S} = (X, \xrightarrow{\Sigma}\leq)$ is *positive* if for every $w \in \Sigma^*$, w is positive for some $x \in X$ if and only if w is positive.

We establish a necessary and sufficient condition for repeated coverability in terms of the stuttering automaton and positive sequences:

▶ **Proposition 17.** *Let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a positive very-WSTS, and let $x, y \in X$. State y is repeatedly coverable from x if and only if there are states c, d of the stuttering automaton $\mathcal{A}_{\downarrow x}$ and $w \in \Sigma^+$ such that $c \xrightarrow{w}_{\mathcal{A}} d$,* $\text{num-accel}(c) = \text{num-accel}(d)$, *$w$ is positive and $y \in \mathrm{ideal}(c)$.*

Proposition 17 allows us to show the decidability of repeated coverability under the following effectiveness hypothesis. A class $\mathcal{C}$ of WSTS is *positive-effective* if there is an

[3] Recall that such a wqo is known as an ω^2-wqo [19]. That we find the ordinal ω^2 in the statement of Theorem 16 and in the notion of ω^2-wqo seems to be coincidental.

algorithm that decides, on input $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq) \in \mathcal{C}$ and a finite automaton A, whether the language of A contains a positive sequence. VAS, Petri nets and ω-Petri nets are positive-effective, since, for these models, testing whether a finite automaton A accepts some positive sequence amounts to computing the Parikh image of $L(A)$, which is effectively semilinear [32].

▶ **Theorem 18.** *Repeated coverability is decidable for completion-post-effective, ∞-completion-effective and positive-effective classes of positive very-WSTS.*

Proof. By Prop. 17, y is repeatedly coverable from x if and only if there are states c, d in $\mathcal{A}_{\downarrow x}$ and $w \in \Sigma^+$ such that:

$$c \xrightarrow{w}_{\mathcal{A}} d, \text{num-accel}(c) = \text{num-accel}(d), w \text{ is positive and } y \in \text{ideal}(c). \tag{1}$$

We show how (1) can be tested. For every $c \in \mathcal{A}_{\downarrow x}$, let A_c be the finite automaton over alphabet Σ whose set of states is $Q_c \stackrel{\text{def}}{=} \{d \in \mathcal{A}_{\downarrow x} : c \xrightarrow{*}_{\mathcal{A}} d \text{ and num-accel}(c) = \text{num-accel}(d)\}$, the initial state is c, all states are accepting, and transitions are as in $\mathcal{A}_{\downarrow x}$. For every $d \in Q_c$ and $w \in \Sigma^*$, $c \xrightarrow{w}_{\mathcal{A}} d$ if and only if w is in the language L_c of A_c. Build a new finite automaton A_c^+ that recognizes $L_c \setminus \{\varepsilon\}$. Let $C_y \stackrel{\text{def}}{=} \{c \in \mathcal{A}_{\downarrow x} : y \in \text{ideal}(c)\}$. By (1), y is repeatedly coverable from x if and only if there exists $c \in C_y$ such that the language $L_c \setminus \{\varepsilon\}$ of A_c^+ contains a positive sequence. The latter is decidable since $\mathcal{C}$ is positive-effective, since A_c^+ can be constructed effectively for every c (because $\mathcal{A}_{\downarrow x}$ can, using the fact that $\mathcal{C}$ is completion-post-effective and ∞-completion-effective), and since we can build C_y by enumerating the states c of $\mathcal{A}_{\downarrow x}$, checking whether $y \in \text{ideal}(c)$ for each (item (2) in the definition of completion-post-effectiveness). ◀

6.2 From model checking to repeated coverability

We conclude this section by reducing LTL model checking to repeated coverability. Recall that a *Büchi automaton* $\mathcal{B}$ is a non-deterministic finite automaton $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ interpreted over Σ^ω. An infinite word is acccepted by $\mathcal{B}$ if it contains an infinite path from q_0 labeled by w and visiting F infinitely often. We denote by $\mathrm{L}(\mathcal{B})$ the set of infinite words accepted by $\mathcal{B}$.

Let $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton and let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a WSTS. The product of $\mathcal{B}$ and $\mathcal{S}$ is defined as $\mathcal{B} \times \mathcal{S} \stackrel{\text{def}}{=} (Q \times X, \xrightarrow{\Sigma \times Q}, = \times \leq)$ where $(p, x) \xrightarrow{(a,r)} (q, y)$ if $(p, a, r) \in \delta, q = r$ and $x \xrightarrow{a} y$. The point in including r in the label is so that $\overbrace{\mathcal{B} \times \mathcal{S}}$ is deterministic, a requirement for very-WSTS. For every WSTS $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$, we extend $\mathcal{S}$ with a new "minimal" element $\bot$ smaller than every other states, i.e. $\mathcal{S}_\bot \stackrel{\text{def}}{=} (X \cup \{x_\bot\}, \xrightarrow{\Sigma}, \leq_\bot)$ where transition relations are unchanged, and $\leq_\bot \stackrel{\text{def}}{=} \leq \cup \{(\bot, y) : y \in X \cup \{\bot\}\}$. It can be shown that if $\mathcal{S}$ is a positive very-WSTS, then $\mathcal{B} \times \mathcal{S}_\bot$ is also. Taking the product of $\mathcal{B}$ and $\mathcal{S}_\bot$ allows us to test whether a word of $\mathrm{L}(\mathcal{B})$ is also an infinite trace of $\mathcal{S}$:

▶ **Proposition 19.** *Let $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ be a Büchi automaton, let $\mathcal{S} = (X, \xrightarrow{\Sigma}, \leq)$ be a very-WSTS, and let $x_0 \in X$. There exists $w \in \mathrm{L}(\mathcal{B}) \cap \omega\text{-Traces}_{\mathcal{S}}(x_0)$ if and only if there exists $q_f \in F$ such that $(q_f, \bot)$ is repeatedly coverable from (q_0, x_0) in $\mathcal{B} \times \mathcal{S}_\bot$.*

Theorem 18 and Proposition 19 imply the decidability of LTL model checking:

▶ **Theorem 20.** *LTL model checking is decidable for completion-post-effective, ∞-completion-effective and positive-effective classes of positive very-WSTS.*

Theorem 20 implies that LTL model checking for ω-Petri nets is decidable. This includes, and generalizes strictly, the decidability of termination in ω-Petri nets [24].

7 Discussion and further work

We have presented the framework of very-WSTS, for which we have given a Karp-Miller algorithm. This allowed us to show that ideal decompositions of coverability sets of very-WSTS are computable, and that LTL model checking is decidable under some additional assumptions. We have also characterized acceleration levels in terms of ordinal ranks. Finally, we have shown that downward traces inclusion is decidable for very-WSTS.

As future work, we propose to study well-structured models beyond very-WSTS for which there exist Karp-Miller algorithms, e.g. unordered data Petri nets (UDPN) [28, 27], or for which reachability is decidable, e.g. recursive Petri nets[4] [26] with strict monotonicity. It is conceivable that LTL model checking is decidable for such models. Our approach will have to be extended to tackle this problem. For example, UDPN do not have finitely many acceleration levels. To circumvent this issue, Hofman et al. [27] make use of two types of accelerations that can be nested. One type is prioritized to ensure that acceleration levels along a branch grow "fast enough" for the algorithm to terminate.

References

1 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 313–321, 1996.

2 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.

3 Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. In *Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 316–327, 1994.

4 Christel Baier, Nathalie Bertrand, and Philippe Schnoebelen. On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems. In *Proc. 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, (LPAR)*, pages 347–361, 2006.

5 Nathalie Bertrand and Philippe Schnoebelen. Computable fixpoints in well-structured symbolic model checking. *Formal Methods in System Design*, 43(2):233–267, 2013.

6 Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching WSTS. In *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 13–25, 2014.

7 Michael Blondin, Alain Finkel, and Pierre McKenzie. Well behaved transition systems. *Logical Methods in Computer Science*, 2017 (accepted).

8 Robert Bonnet. On the cardinality of the set of initial intervals of a partially ordered set. In *Infinite and finite sets: to Paul Erdős on his 60^{t}extth birthday*, pages 189–198. North-Holland, 1975.

9 Pierre Chambart, Alain Finkel, and Sylvain Schmitz. Forward analysis and model checking for trace bounded WSTS. In *Proc. 32nd International Conference on Applications and Theory of Petri Nets*, 2011.

10 Pierre Chambart, Alain Finkel, and Sylvain Schmitz. Forward analysis and model checking for trace bounded WSTS. *Theor. Comput. Sci.*, 637:1–29, 2016.

[4] Recursive Petri nets are WSTS for the tree embedding.

11 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th International Colloquium Automata, Languages and Programming (ICALP)*, pages 103–115, 1998.

12 E. Allen Emerson and Kedar S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc. 13th IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–80, 1998.

13 Paul Erdős and Alfred Tarski. On families of mutually exclusive sets. *Annals of Mathematics*, 2(44):315–329, 1943.

14 Javier Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. 19th International Colloquium on Trees in Algebra and Programming (CAAP)*, pages 115–129, 1994.

15 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 352–359, 1999.

16 Alain Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *Proc. 14th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 499–508, 1987.

17 Alain Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

18 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In *STACS'09*, pages 433–444, Freiburg, Germany, 2009. Leibniz-Zentrum für Informatik, Intl. Proc. in Informatics 3.

19 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part II: complete WSTS. *Logical Methods in Computer Science*, 8(3), 2012.

20 Alain Finkel, Pierre McKenzie, and Claudine Picaronny. A well-structured framework for analysing Petri net extensions. *Information and Computation*, 195(1-2):1–29, 2004.

21 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

22 Thomas Forster. A tutorial on countable ordinals. Available from the Web at `https://www.dpmms.cam.ac.uk/~tf/fundamentalsequence.pdf`, 2010. Read on Feb. 03, 2017.

23 Roland Fraïssé. Theory of relations. *Studies in Logic and the Foundations of Mathematics*, 118:1–456, 1986.

24 Gilles Geeraerts, Alexander Heußner, M. Praveen, and Jean-François Raskin. ω-Petri nets: Algorithms and complexity. *Fundamenta Informaticae*, 137(1):29–60, 2015.

25 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *J. Comput. Syst. Sci.*, 72(1):180–203, 2006. `doi:10.1016/j.jcss.2005.09.001`.

26 Serge Haddad and Denis Poitrenaud. Recursive Petri nets. *Acta Inf.*, 44(7-8):463–508, 2007.

27 Piotr Hofman, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for Petri nets with unordered data. In *FoSSaCS*, pages 445–461, 2016.

28 Reiner Hüchting, Rupak Majumdar, and Roland Meyer. Bounds on mobility. In *Proc. 25th International Conference on Concurrency Theory (CONCUR)*, pages 357–371, 2014.

29 Richard M. Karp and Raymond E. Miller. Parallel program schemata: a mathematical model for parallel computation. In *Proc. 8th Annual Symposium on Switching and Automata Theory*, pages 55–61. IEEE Computer Society, 1967.

30 E. V. Kouzmin, Nikolay V. Shilov, and Valery A. Sokolov. Model checking mu-calculus in well-structured transition systems. In *Proc. 11th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 152–155, 2004.

31 J.D. Lawson, M. Mislove, and H. Priestley. Ordered sets with no infinite antichains. *Discrete Mathematics*, 63(2):225–230, 1987.

32 Rohit J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

33 Maurice Pouzet. Relations non reconstructibles par leurs restrictions. *Journal of Combinatorial Theory, Series B*, 26(1):22–34, 1979.

34 Maurice Pouzet and Nejib Zaguia. Dimension de Krull des ensembles ordonnés. *Discrete Mathematics*, 53:173–192, 1985.

35 Fernando Rosa-Velardo and María Martos-Salgado. Multiset rewriting for the verification of depth-bounded processes with name binding. *Inf. Comput.*, 215:68–87, 2012.

36 Fernando Rosa-Velardo, María Martos-Salgado, and David de Frutos-Escrig. Accelerations for the coverability set of Petri nets with names. *Fundamenta Informaticae*, 113(3-4):313–341, 2011.

37 Philippe Schnoebelen. Lossy counter machines decidability cheat sheet. In *Proc.* 4^{th} *International Workshop on Reachability Problems (RP)*, pages 51–75, 2010.

38 Rüdiger Valk. Self-modifying nets, a natural extension of Petri nets. In *Proc.* 5^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–476, 1978.

39 Rüdiger Valk and Matthias Jantzen. The residue of vector sets with applications to decidability problems in Petri nets. *Acta Inf.*, 21:643–674, 1985.

40 Kumar N. Verma and Jean Goubault-Larrecq. Karp-Miller trees for a branching extension of VASS. *Discrete Mathematics & Theoretical Computer Science*, 7(1):217–230, 2005.

41 Damien Zufferey, Thomas Wies, and Thomas A. Henzinger. Ideal abstractions for well-structured transition systems. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 445–460. Springer, 2012. doi:10.1007/978-3-642-27940-9_29.

Rabin vs. Streett Automata*

Udi Boker

Interdisciplinary Center (IDC), Herzliya, Israel

Abstract

The Rabin and Streett acceptance conditions are dual. Accordingly, deterministic Rabin and Streett automata are dual. Yet, when adding nondeterminsim, the picture changes dramatically. In fact, the state blowup involved in translations between Rabin and Streett automata is a longstanding open problem, having an exponential gap between the known lower and upper bounds.

We resolve the problem, showing that the translation of Streett to Rabin automata involves a state blowup in $\Theta(n^2)$, whereas in the other direction, the translations of both deterministic and nondeterministic Rabin automata to nondeterministic Streett automata involve a state blowup in $2^{\Theta(n)}$.

Analyzing this substantial difference between the two directions, we get to the conclusion that when studying translations between automata, one should not only consider the state blowup, but also the *size* blowup, where the latter takes into account all of the automaton elements. More precisely, the size of an automaton is defined to be the maximum of the alphabet length, the number of states, the number of transitions, and the acceptance condition length (index).

Indeed, size-wise, the results are opposite. That is, the translation of Rabin to Streett involves a size blowup in $\Theta(n^2)$ and of Streett to Rabin in $2^{\Theta(n)}$. The core difference between state blowup and size blowup stems from the tradeoff between the index and the number of states. (Recall that the index of Rabin and Streett automata might be exponential in the number of states.)

We continue with resolving the open problem of translating deterministic Rabin and Streett automata to the weaker types of deterministic co-Büchi and Büchi automata, respectively. We show that the state blowup involved in these translations, when possible, is in $2^{\Theta(n)}$, whereas the size blowup is in $\Theta(n^2)$.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Finite automata on infinite words, translations, automata size, state space

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.17

1 Introduction

Automata on infinite words were introduced in the 60's, in the course of solving fundamental decision problems in mathematics and logic [4, 19, 15, 21]. Today, they are widely used in formal verification and synthesis of nonterminating systems, where their size and the complexity of performing operations on them play a key role. Unlike automata on finite words, there are several types of automata on infinite words, differing in their acceptance conditions, most notably Büchi [4], Muller [19], Rabin [21], Streett [27], and parity [18]. Each of the types has its own advantages, for which reason there is an extensive research on the state blowup involved in the translations between them [22, 23, 12, 20, 28, 9, 13, 25, 2, 26, 1].

* This work was supported by the Israel Science Foundation grant 1373/16.

For most translations, there are satisfactory solutions, in the sense that the upper bound on the state blowup involved in the translation algorithm is close to the theoretical lower bound on the inevitable blowup. For some stubborn cases, however, there is still an exponential gap between the lower and upper bounds. This situation is especially frustrating, as it implies that not only is something missing in our understanding of automata on infinite words, but also that we may be using algorithms that can be significantly improved.

Most of these stubborn cases concern Rabin and Streett automata. In particular, the best known algorithm for translating deterministic and nondeterministic Streett automata to nondeterministic Rabin automata involves a $2^{O(n)}$ state blowup [8, 22], while the current lower bound is only $\Omega(n)$. As for the other direction, one can deduce an exponential state blowup in the translation of nondeterministic Rabin to Streett automata, due to the doubly exponential blowup in determinizing Rabin automata [7] and the singly exponential blowup in determinizing Streett automata [5]. Yet, for the translation of deterministic Rabin to nondeterministic Streett automata, there is currently also an $\Omega(n)$ lower bound and a $2^{O(n)}$ upper bound [8].

We resolve these problems, providing tight bounds for both directions. Interestingly, we show that the translation of Streett to Rabin automata involves a state blowup in $\Theta(n^2)$, whereas the translations of both deterministic and nondeterministic Rabin automata to nondeterministic Streett automata involve a state blowup in $2^{\Theta(n)}$.

For the translation of Streett to Rabin automata, we provide in Section 3 a new algorithm. Given a Streett automaton with n states, the constructed Rabin automaton has up to $2n^2$ states. We couple it with a lower bound proof, showing that a quadratic state blowup is optimal.

The challenge in translating Streett to Rabin comes from the conjunctive nature of the former and the disjunctive nature of the latter. That is, by the Streett acceptance condition, one can require, for example, to visit each of n states infinitely often, whereas the Rabin condition allows to ask for infinitely many visits in at least one of the n states.

The standard solution to require with a Rabin condition a visit in each of n different states is to have n copies of the original automaton and allow a move from the i-th copy to the next one only upon visiting the i-th state [8, 22]. A Streett acceptance condition on an automaton with n states might induce a choice between $2^{O(n)}$ different requirements to visit each of $O(n)$ different states, implying that the resulting Rabin automaton has up to $n2^{O(n)} = 2^{O(n)}$ states.

Our construction allows to use the same n copies for all of the $2^{O(n)}$ requirements. The idea is to add "bridges" between each two such copies, and provide a Rabin acceptance pair $\langle B, G\rangle$ for each requirement, such that the "bad" set B of the Rabin condition forces a transition through the bridge only when the relevant state is visited.

For the other direction, we provide in Section 4 a $2^{\Omega(n)}$ lower bound on the state blowup involved in the translation of deterministic Rabin to nondeterministic Streett automata. The lower bound builds on the property of the Streett condition, according to which the union of two accepting cycles is accepting. (By a "union of cycles" we mean a cycle whose states are the union of the states of the two cycles.) We describe a family $\{\mathcal{D}_n\}_{n\geq 1}$ of deterministic Rabin automata, and for each automaton $\mathcal{D}_n$, a set of $2^{\Omega(n)}$ words, such that $\mathcal{D}_n$ accepts each of the words, but none of their combinations. We then show that each such word can be associated with a unique state of a Streett automaton equivalent to $\mathcal{D}_n$.

Upfront, the bold asymmetry of the state blowup involved in the two directions is very surprising. Yet, a close look on the lower and upper bound results reveals the reason—there is a tradeoff between the number of states and the acceptance condition length (index). In

the translation of Rabin to Streett there is an exponential state blowup and no index blowup, whereas in the translation of Streett to Rabin there is a quadratic state blowup and an exponential index blowup.

We thus argue that when studying translations between automata, one should not only consider the state blowup, but also the *size* blowup, where the latter takes into account all of the automaton elements. More precisely, the size of an automaton is defined to be the maximum of the alphabet length, the number of states, the number of transitions, and the index. There are literature results that take the index blowup into account, for example [24], but it is often not the case.

Out of the four elements that constitute the automaton size, the number of states and the index are the dominant ones. Considering the alphabet, the common practice is to provide the upper bounds for arbitrary alphabets and to seek lower bounds with fixed alphabets. For example, [14] strengthen the lower bound of [16] by moving to a fixed alphabet, and [28] starts with automata over a rich alphabet and then moves to a fixed alphabet. As for the number of transitions, they are bounded by the size of the alphabet times quadratically the number of states, and the transition blowup tends to go hand in hand with the state blowup.

The state and size blowups involved in the translations between Rabin and Streett automata are summarized in Table 1. The differences between the results that concern the state blowup and those that concern the size blowup stem from the fact that the index of Rabin and Streett automata might be exponential in the number of states.

Next, we look into the translations of deterministic Rabin and Streett automata to the weaker types of deterministic co-Büchi and Büchi automata, respectively. It is known that a deterministic Rabin automaton that has an equivalent deterministic Büchi automaton has one on its own structure, namely an equivalent Büchi automaton exists over the same states and transitions [11]. Yet, for the translation of deterministic Rabin to deterministic co-Büchi automata, the upper bound on both the state and size blowups is $2^{O(n)}$ [2] with only an $\Omega(n)$ lower bound. We show that the state blowup of this translation is in $2^{\Theta(n)}$ and that the size blowup is in $\Theta(n^2)$. (The same holds for the dual Streett to Büchi case.)

To this end, we provide a new algorithm for translating, when possible, a deterministic Rabin automaton with n states and index k to a deterministic co-Büchi automaton with nk states. The translation goes through an intermediate nondeterministic co-Büchi automaton, as per the constriction in [2]. We analyze the intermediate automaton to be of a special form, having nk states. We then use its special form for determinizing it over the same structure.

In all of our results, whenever possible, we also consider the translations of automata with the more descriptive Muller condition.

2 Preliminaries

Given a finite alphabet Σ, a *word* over Σ is a (possibly infinite) sequence $w = w(0) \cdot w(1) \cdots$ of letters in Σ.

An *automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and α is an acceptance condition. The first four elements, namely $\langle \Sigma, Q, \delta, Q_0 \rangle$, are the automaton's *structure*. The automaton $\mathcal{A}$ may have several initial states and the transition function may specify many possible transitions for each state and letter, and hence we say that $\mathcal{A}$ is *nondeterministic*. In the case where $|Q_0| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\delta(q, \sigma)| \leq 1$, we say that $\mathcal{A}$ is *deterministic*. We then use q_0 instead of Q_0 to denote the single initial state. For a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the automaton that is derived

Table 1 The blowup involved in the translations between Rabin and Streett automata.

State Blowup

From \ To	Rabin	
	Det.	Non-Det.
Det. Streett	$\Theta(2^{n \log n})$ [10, 14]	$\Theta(n^2)$ Thm. 1, 2, Cor. 3
Non-Det. Streett	$2^{\Theta(n^2 \log n)}$ [6, 5]	

From \ To	Streett	
	Det.	Non-Det.
Det. Rabin	$\Theta(2^{n \log n})$ [10, 14]	$2^{\Theta(n)}$ Thm. 4, Cor. 5
Non-Det. Rabin	$2^{2^{\Theta(n)}}$ [22, 7]	

Size Blowup

From \ To	Rabin	
	Det.	Non-Det.
Det. Streett	$\Theta(2^{n \log n})$ [10, 14]	$2^{\Theta(n)}$ [8, 24]
Non-Det. Streett	$2^{\Theta(n^2 \log n)}$ [6, 5]	

From \ To	Streett	
	Det.	Non-Det.
Det. Rabin	$\Theta(2^{n \log n})$ [10, 14]	$O(n^2)$ [8]
Non-Det. Rabin	$2^{\Theta(n^2 \log n)}$ [22, 7]	

from $\mathcal{A}$ by changing the set of initial states to $\{q\}$.

A *run*, or a *path*, $r = r(0), r(1), \cdots$ of $\mathcal{A}$ on $w = w(0) \cdot w(1) \cdots \in \Sigma^\omega$ is an infinite sequence of states such that $r(0) \in Q_0$, and for every $i \geq 0$, we have $r(i+1) \in \delta(r(i), w(i))$.

Acceptance is defined with respect to the set $inf(r)$ of states that the run r visits infinitely often. Formally, $inf(r) = \{q \in Q \mid \text{ for infinitely many } i \in \mathbb{N}, \text{ we have } r(i) = q\}$. As Q is finite, it is guaranteed that $inf(r) \neq \emptyset$.

Several acceptance conditions are studied in the literature; the main ones are:

- *Büchi*, where $\alpha \subseteq Q$, and r is accepting iff $inf(r) \cap \alpha \neq \emptyset$. (The states of α are *accepting*.)
- *co-Büchi*, where $\alpha \subseteq Q$, and r is accepting iff $inf(r) \cap \alpha = \emptyset$. (The states of α are *rejecting*.)
- *weak* is a special case of the Büchi condition, where every strongly connected component of the automaton is either contained in α or disjoint to α; that is, no strongly connected component has a state in α and some other state not in α.
- *parity*, where $\alpha = \{S_1, S_2, \ldots, S_{2k}\}$ with $S_1 \subset S_2 \subset \cdots \subset S_{2k} = Q$, and r is accepting if the minimal index i for which $inf(r) \cap S_i \neq \emptyset$ is even.
- *Rabin*, where $\alpha = \{\langle B_1, G_1\rangle, \langle B_2, G_2\rangle, \ldots, \langle B_k, G_k\rangle\}$, with $B_i, G_i \subseteq Q$ and r is accepting iff for some $i \in [1..k]$, we have $inf(r) \cap B_i = \emptyset$ and $inf(r) \cap G_i \neq \emptyset$.
- *Streett*, where $\alpha = \{\langle B_1, G_1\rangle, \langle B_2, G_2\rangle, \ldots, \langle B_k, G_k\rangle\}$, with $B_i, G_i \subseteq Q$ and r is accepting iff for all $i \in [1..k]$, we have $inf(r) \cap B_i = \emptyset$ or $inf(r) \cap G_i \neq \emptyset$.
- *Muller*, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, with $\alpha_i \subseteq Q$ and r is accepting iff for some $i \in [1..k]$, we have $inf(r) = \alpha_i$.

A run that is not accepting is *rejecting*. Notice that Büchi and co-Büchi are special cases of the parity condition, which is in turn a special case of both the Rabin and Streett conditions. In the latter conditions, we refer to the B_i and G_i sets as the "bad" and "good" sets, respectively.

The number of sets in the parity and Muller acceptance conditions or pairs in the Rabin and Streett acceptance conditions is called the *index* of the automaton. For weak, co-Büchi, and Büchi automata, the index is 1.

The *size* of an automaton is the maximum size of its elements; more precisely, it is the maximum of the alphabet length, the number of states, the number of transitions, and the index.

An automaton accepts a word if it has an accepting run on it. The language of an automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We also say that $\mathcal{A}$ *recognizes* the language $L(\mathcal{A})$. Two automata, $\mathcal{A}$ and $\mathcal{A}'$, are *equivalent* iff $L(\mathcal{A}) = L(\mathcal{A}')$.

For a finite path $C = q_1 q_2 \cdots q_n$, we say that C is accepting (resp., rejecting) if the infinite path C^ω is accepting (resp., rejecting). Notice that the union of two Rabin-rejecting (finite) paths is Rabin-rejecting, and of two Streett-accepting (finite) paths is Streett-accepting.

The *class* of an automaton characterizes its transition mode (deterministic or nondeterministic) and its acceptance condition. In the more technical paragraphs, we shall denote the different classes of automata by three letter acronyms in {D, N} × {W, B, C, P, R, S, M} × {W}. The first letter stands for the transition mode of the automaton (deterministic or nondeterministic); the second for the acceptance-condition (weak, Büchi, co-Büchi, parity, Rabin, Streett, or Muller); and the third indicates that the automaton runs on words. For example, DBW stands for deterministic Büchi automata on words.

Büchi, parity, Rabin, Streett, and Muller automata have the same expressive power, recognizing all ω-regular languages. Weak and co-Büchi automata, as well as deterministic Büchi automata, are less expressive. When an automaton $\mathcal{A}$ of type γ has an equivalent automaton of type γ', we say that $\mathcal{A}$ is γ'-*recognizable*, for example NCW-recognizable.

3 Streett to Rabin

In this section we consider the translation of Streett to Rabin automata. The best known algorithm involves a $2^{O(n)}$ state blowup [8, 22], while the current lower bound is only $\Omega(n)$. We provide a new translation algorithm that involves a $2n^2$ state blowup, and show that a quadratic blowup is optimal. We then analyze the size blowup, and show that very differently from the state blowup it is exponential.

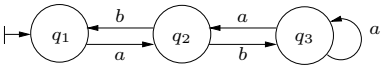
State blowup

We show that every Muller automaton can be translated to a Rabin automaton with only a quadratic state blowup, implying the result for translating Streett to Rabin. We start with an informal explanation of the construction, followed by an illustrated example and a formal proof.

Consider an automaton structure A with n states, and a Muller acceptance set S. A run r is accepting according to S if it visits infinitely often all the states in S and only finitely often the states out of S. Let us look first what can be done with a Rabin automaton that is defined over A. We can easily define a Rabin acceptance pair $\langle B, G\rangle$ that partially corresponds to the Muller acceptance set S—We define B to include all the states out of S, ensuring that they are visited only finitely often. The problem is that the set G cannot force visits in all states of S; It can only force a visit in some states of S.

In order to force a visit in every state of a set $S = \{q_1, q_2, \ldots, q_{|S|}\}$, one can take $|S|$ copies of A (which we call "components"), move from the i-th component to the next one (modulo $|S|$) upon reaching the state q_i, and setting, say, q_1 in the first component to be the only accepting state. This is, for example, the idea in translating a generalized Büchi

A Muller automaton $\mathcal{A}$:



The Muller acceptance sets:

i) $\{q_1, q_3\}$

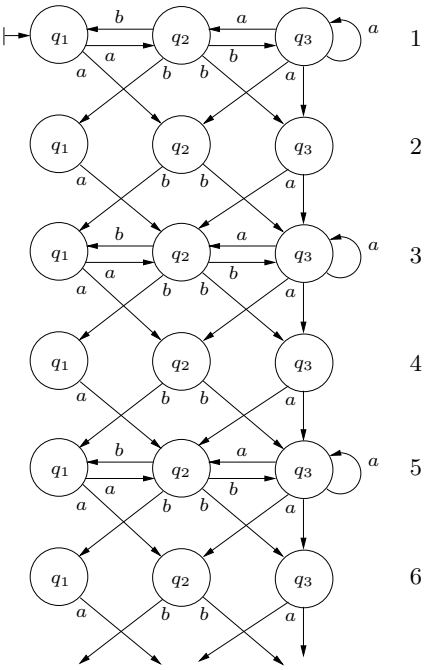
ii) $\{q_2, q_3\}$

The corresponding Rabin acceptance pairs:

i) $\langle B, G\rangle$, where
$G = \{\langle q_1, 2\rangle\}$
$B = B_1 \cup B_2$, with
$B_1 = \{\langle q_2, 1\rangle, \langle q_2, 2\rangle, \langle q_2, 3\rangle, \langle q_2, 4\rangle, \langle q_2, 5\rangle, \langle q_2, 6\rangle\}$
$B_2 = \{\langle q_2, 2\rangle, \langle q_3, 2\rangle, \langle q_1, 6\rangle, \langle q_2, 6\rangle\}$

ii) $\langle B, G\rangle$, where
$G = \{\langle q_2, 4\rangle\}$
$B = B_1 \cup B_2$, with
$B_1 = \{\langle q_1, 1\rangle, \langle q_1, 2\rangle, \langle q_1, 3\rangle, \langle q_1, 4\rangle, \langle q_1, 5\rangle, \langle q_1, 6\rangle\}$
$B_2 = \{\langle q_1, 4\rangle, \langle q_3, 4\rangle, \langle q_1, 6\rangle, \langle q_2, 6\rangle\}$

An equivalent Rabin autmaton $\mathcal{A}'$:



To component no. 1

Figure 1 An example of the translation of a Muller automaton to an equivalent Rabin automaton with an $O(n^2)$ state blowup, following the construction in the proof of Theorem 1.

automaton to a Büchi automaton. (The generalized-Büchi acceptance condition allows for several sets of states, and a run is accepting if it visits infinitely often each of these sets.)

The problem with the above approach is that we need $|S|$ copies of A for every Muller acceptance set S. As there might be 2^n Muller acceptance sets, we get an exponential state blowup. This blowup is indeed inevitable when translating to a Büchi automaton [24], yet it is not clear if and how the Rabin acceptance condition can help. We show that it certainly can, allowing all the Muller sets S to be handled over the same set of components.

We extend the above approach of having copies of A, by adding a "bridge" between each two copies. A bridge is a limited copy of A, in which all states can only move to the next component. Then, for every Muller acceptance set S, we define a Rabin acceptance pair $\langle B, G\rangle$ that forces for every state $q_i \in S$, a visit to the state q_i of the i-th bridge—We add to B all the states of the i-th bridge, except for q_i. In bridges of a component j, such that $q_j \notin S$, the run can visit any state of S.

An example of a translation of a Muller to Rabin automaton along this construction is illustrated in Figure 1.

▶ **Theorem 1.** *For every NMW with n states, m transitions, and index k, there is an equivalent NRW with $2n^2$ states, $3nm$ transitions, and index k.*

Proof.

Construction. Consider an NMW $\mathcal{A} = \langle \Sigma, Q = \{q_1, q_2, \ldots, q_n\}, Q_0, \delta, \alpha\rangle$ with n states, m transitions, and index k. We define the NRW $\mathcal{A}' = \langle \Sigma, Q', Q'_0, \delta', \alpha'\rangle$, which we claim to be equivalent to $\mathcal{A}$, as follows.

- $Q' = Q \times [1..2n]$. (We shall call each instance of Q a "component" of $\mathcal{A}'$.)
- $Q'_0 = Q_0 \times \{1\}$.
- For every state $\langle q, j\rangle \in Q'$ and $\sigma \in \Sigma$, the transition function is defined as follows.
  - If j is odd, then $\delta'(\langle q, j\rangle, \sigma) = \{\langle \hat{q}, \hat{j}\rangle \mid \hat{q} \in \delta(q, \sigma) \text{ and } \hat{j} \in \{j, j{+}1\}\}$.
  - If j is even, then $\delta'(\langle q, j\rangle, \sigma) = \{\langle \hat{q}, (j{+}1) \bmod 2n\rangle \mid \hat{q} \in \delta(q, \sigma)\}$.
- For every Muller accepting set $S \in \alpha$, we have in α' the Rabin acceptance pair $\langle B, G\rangle$, where B and G are defined as follows. Let $x \in [1..n]$ be the minimal index i of a state $q_i \in S$.
  - $G = \{\langle q_x, 2x\rangle\}$ consists of the single state q_x in the $2x$ component.
  - B is the union of two sets B_1 and B_2. The first includes all the states that are not in S, along all the components. The second handles the transitions through the even components (the bridges), adding every state $q_i \in S$ that appears in a component j, such that $j \neq i$ and $q_j \in S$. Formally, $B_1 = \{\langle q, j\rangle \mid q \notin S \text{ and } j \in [1..2n]\}$, and $B_2 = \{\langle q_i, 2j\rangle \mid i \neq j \in [1..n] \text{ and } q_j \in S\}$.

Correctness. Consider a word $w \in L(\mathcal{A})$. Then there is a run r of $\mathcal{A}$ and a set $S \in \alpha$, such that r visits infinitely often exactly the states in S. We will describe a run r' of $\mathcal{A}'$ that satisfies the Rabin acceptance pair $\langle B, G\rangle$ that corresponds to S. The left-projection of r', namely the q_i element of the $\langle q_i, j\rangle$ states that r' visits, is identical to r. We will describe the right-projection of r', namely the series of components that r' traverses along the run.

Notice that when r' is in an even component it must move to the next component, and when it is in an odd component it has the choice of whether to stay there or move to the next one. We explain next how r' behaves in the odd components. Let t be the first position of r after which it only visits states in S. The run r' remains in the first component until position t. After position t, when r' is in component $2j-1$, it remains there until one of the following happens: i) the next state of r is q_j and $q_j \in S$; or ii) the next state of r is q_x and $q_j \notin S$. Note that one of the above events must indeed eventually happen: r must eventually visit q_x, because $q_x \in S$, and in the case that $q_j \in S$, r must also eventually visit q_j.

Observe that r' satisfies the Rabin acceptance pair $\langle B, G\rangle$: Considering the "good" set $G = \{\langle q_x, 2x\rangle\}$, r' visits all the components infinitely often, and when in component $2x$, it visits the state q_x. Considering the "bad" set $B = B_1 \cup B_2$, the states of B_1 are visited only finitely often, since the left-projection of r' is identical to r. As for B_2, its states are visited only finitely often, since the only case in which r' visits after position t a state $\langle q_i, 2j\rangle$ such that $i \neq j$ is when $q_j \notin S$.

As for the other direction, consider a word $w \in L(\mathcal{A}')$. Then there is a run r' of $\mathcal{A}'$ that satisfies some Rabin acceptance pair $\langle B, G\rangle$ of $\mathcal{A}'$. By the construction of $\mathcal{A}'$, the pair $\langle B, G\rangle$ corresponds to some Muller set $S \in \alpha$. We claim that the left-projection of r' is a run r of $\mathcal{A}$ that visits infinitely often exactly the states in S.

First observe that due to the subset B_1 of B, the run r visits states out of S only finitely often. Next, observe that r' must visit infinitely often all components—it visits $\{\langle q_x, 2x\rangle\}$ infinitely often, and going from $\{\langle q_x, 2x\rangle\}$ back to itself enforces a visit in all components. Now, by the subset B_2 of B, when r' is in component $2j$ and $q_j \in S$, it can visit finitely often only a state different from $\langle q_j, 2j\rangle$. Hence, r' visits infinitely often $\langle q_j, 2j\rangle$, for every $q_j \in S$, and therefore r visits infinitely often all states in S. ◀

Next, we provide a matching lower bound. As opposed to the upper bound construction, a lower bound on the state blowup involved in the translation of Muller to Rabin automata does not hold for the translation of Streett automata, as the Streett condition is less descriptive

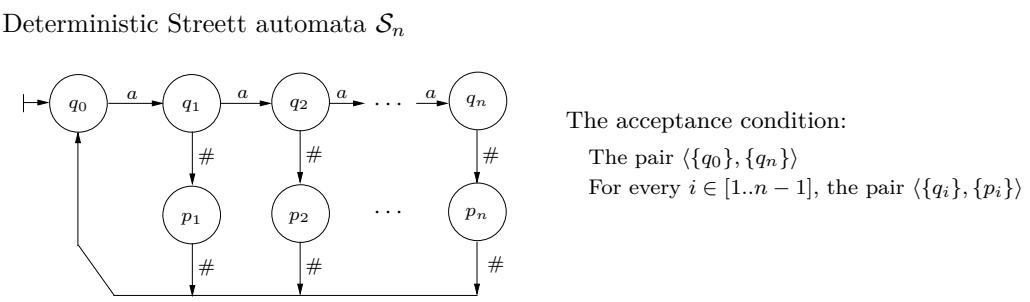


Figure 2 Deterministic Streett automata $\mathcal{S}_n$ with $O(n)$ states, for which equivalent nondeterministic Rabin automata have at least $n^2/2$ states.

than the Muller one. Yet, it turns out that the family of languages used in [1] for the former, can also serve us for the latter (and even for translating generalized Büchi automata).

▶ **Theorem 2.** *For every $n \in \mathbb{N}$, there is a DSW over a two-letter alphabet with $2n+1$ states, $3n$ transitions, and index n, for which equivalent NRWs have at least $n^2/2$ states.*

Proof. Consider the DSWs $\mathcal{S}_n$ depicted in Figure 2. Observe that a run of $\mathcal{S}_n$ is accepting iff it visits all of $\mathcal{S}_n$'s states infinitely often. Indeed, every run must visit q_0 infinitely often, and by the first acceptance pair, it must also visit q_n infinitely often. Every visit to q_n entails a visit to p_n and to q_i, for all $i \in [1..n-1]$, which in turn entail, by the rest of the acceptance pairs, a visit to p_i, for all $i \in [1..n-1]$.

Hence, $\mathcal{S}_n$ is equivalent to a Muller automaton $\mathcal{M}_n$ over the same structure with a single acceptance set that includes all of $\mathcal{M}_n$'s states. It is shown in [1, Proof of Theorem 9] that every NRW equivalent to $\mathcal{M}_n$ has at least $n^2/2$ states. ◀

We conclude with the corresponding tight bounds.

▶ **Corollary 3.** *The state blowup in the translations of deterministic and nondeterministic Streett and Muller automata to nondeterministic Rabin automata is in $\Theta(n^2)$.*

Proof. The upper bounds follow from Theorem 1. The lower bound for Streett from Theorem 2 and for Muller from [1, Proof of Theorem 9] . ◀

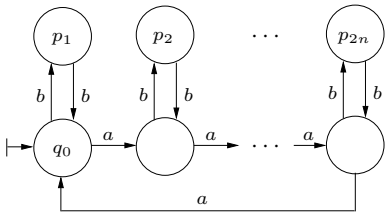
Size blowup

The size blowup involved in translations of Streett to Rabin automata is very different from the state blowup, as it is exponential, even when considering a deterministic Streett automaton: In [22, Lemma 2.3], there is a family of languages L_n, for $n > 0$, over a fixed alphabet, such that L_n is recognized by a DSW with $O(n)$ states, transitions, and index, while an equivalent NBW requires $\Omega(2^n)$ states. As every NRW with n states and index k can be translated to an equivalent NBW with nk states [8], it follows that the size blowup in translating DSW to NRW is in $2^{\Omega(n)}$, and together with known constructions [24], it is in $2^{\Theta(n)}$.

4 Rabin to Streett

In this section we consider the translation of Rabin to Streett automata. It turns out that the state and size blowups in this case "switch roles" with the corresponding blowups in the translation of Streett to Rabin—the size blowup is known to be quadratic [8], while

Deterministic Rabin automata $\mathcal{D}_n$



The acceptance condition:

Let Q be the set of all states and $P = \{p_1, p_2, \ldots, p_{2n}\}$.

The acceptance pairs are $\{\langle B, Q\rangle \mid B \subseteq P \text{ and } |B| = n\}$.

Figure 3 Deterministic Rabin automata with $O(n)$ states, for which equivalent nondeterministic Streett automata need at least 2^n states.

we provide an exponential lower bound on the state blowup involved in the translation of deterministic Rabin to nondeterministic Streett automata.

State blowup

Our lower bound proof for the state blowup involved in the translation of Rabin to Streett automata builds on the property of the Streett condition, according to which the union of two accepting cycles is accepting. The challenge is to come up with a family $\{\mathcal{D}_n\}_{n\geq 1}$ of deterministic Rabin automata, and for each such automaton a set of $2^{\Omega(n)}$ words, such that $\mathcal{D}_n$ accepts each of the words, but none of their combinations.

We describe such a family in Figure 3. The automaton $\mathcal{D}_n$ accepts words on which it visits finitely often at least n out of the $2n$ states of the set P. We then define a set of $\binom{2n}{n}$ periodic words on which $\mathcal{D}_n$ visits finitely often *exactly* n states of P. Each word in the set corresponds to a choice of n out of the $2n$ states of P. (Recall that $\binom{2n}{n} > 2^n$.)

The repeated finite word in each such infinite word corresponds to a cycle of $\mathcal{D}_n$ from q_0 back to itself, avoiding the relevant n states of P and visiting the other n states of P. As a result, an infinite word that combines two such different finite words is rejected by $\mathcal{D}_n$, as the run of $\mathcal{D}_n$ on it visits finitely often less than n states of P. Accordingly, we can show that for an equivalent Streett automaton, accepting runs on different such words cannot share the same state in positions that start the repeated finite word. Hence, the Streett automaton has at least $\binom{2n}{n}$ different states.

▶ **Theorem 4.** *For every $n \in \mathbb{N}$, there is a DRW over a two-letter alphabet with $4n$ states and $6n$ transitions, for which equivalent NSWs have at least 2^n states.*

Proof. Consider the family $\{\mathcal{D}_n\}_{n\geq 1}$ of DRWs depicted in Figure 3, and let $\mathcal{A}$ be an NSW equivalent to $\mathcal{D}_n$. Observe that $\mathcal{D}_n$ has an index $k = \binom{2n}{n} > 2^n$. We show that $\mathcal{A}$ has a unique state for every acceptance pair of $\mathcal{D}_n$, implying that it has more than 2^n states.

For every $i \in [1..k]$, let B_i be the "bad" (left) set in the i-th acceptance pair of $\mathcal{D}_n$, and let u_i be the minimal finite word that takes $\mathcal{D}_n$ from q_0 back to q_0, while avoiding the states in B_i and visiting (once) every state in $P \setminus B_i$. For example, for $n = 3$ and $B_i = \{2, 5, 6\}$, we have $u_i = bbaabbabbaaa$.

For every $i \in [1..k]$, define $w_i = u_i^\omega$, and notice that $\mathcal{D}_n$ accepts w_i due to its i-th acceptance pair. We shall call the positions of w_i in which $\mathcal{D}_n$ reaches q_0 "big positions". (These are the positions of w_i after every full instance of u_i.) Let r_i be an accepting run of $\mathcal{A}$ on w_i. We now show that for every $i \neq j \in [1..k]$, the states that r_i and r_j visit infinitely often in big positions are disjoint.

Assume toward contradiction that for some $i \neq j \in [1..k]$, both r_i and r_j visit the same state s infinitely often in big positions. Let t and t' be big positions of w_i in which r_i visits s, and between which r_i visits exactly the states that it visits infinitely often. Let u be the subword of w_i between positions t and t'. Now, let w be the word that is derived from w_j by adding u in every big position in which r_j visits s.

Consider the run r of $\mathcal{A}$ on w that follows r_j, while making extra cycles from s back to itself in every big position that u was added to w. In these extra cycles, r follows the cycle that r_i does between positions t and t'. Notice that the states that r visits infinitely often are the union of the states that r_i and r_j visit infinitely often. Hence, due to the property of the Streett condition that the union of two accepting cycles is accepting, we have that r is accepting.

On the other hand, when $\mathcal{D}_n$ runs on w, it reads infinitely often both u_i and u_j from the state q_0, implying that it visits infinitely often both $P \setminus B_i$ and $P \setminus B_j$. Thus, it visits finitely often less than n states in P, and therefore rejects w. Contradiction. ◀

As the Rabin condition is less detailed than the Muller condition (namely, every Rabin automaton as an equivalent Muller automaton over the same structure), and the Streett condition is more detailed than the parity and Büchi conditions, the above results can be generalized as follows.

▶ **Corollary 5.** *The translations of deterministic Rabin and Muller automata to nondeterministic Büchi, parity, and Streett automata involve a state blowup in* $2^{\Theta(n)}$.

Proof. The lower bounds follow from Theorem 4. The upper bounds for translating Rabin automata are given in [8], and for Muller automata are folklore (see [1] for details). ◀

Size blowup

Every Rabin automaton with n states and index k can be translated to an equivalent Büchi automaton with nk states [8], which can also be viewed as a Streett automaton, providing a quadratic size blowup.

5 Rabin to Co-Büchi

In this section we resolve the open problems of translating deterministic Rabin and Streett automata to the weaker types of deterministic co-Büchi and Büchi automata, respectively. For both the state and size blowups, the known upper bound is in $2^{O(n)}$ [2] with only an $\Omega(n)$ lower bound. We show that the state blowup of these translations is in $2^{\Theta(n)}$ and that the size blowup is in $\Theta(n^2)$.

State blowup

Upfront, the lower bound for the translation of Rabin to Streett automata (Theorem 4) does not follow to the case of translating to a deterministic co-Büchi automaton, as the latter does not recognize all ω-regular languages. Yet, our family $\mathcal{D}_n$ of NRWs, as depicted in Figure 3, is NCW-recognizable, providing a lower bound also for the co-Büchi case.

▶ **Theorem 6.** *For every $n \in \mathbb{N}$, there is an NCW-recognizable DRW over a two-letter alphabet with $4n$ states and $6n$ transitions, for which equivalent NCWs have at least 2^n states.*

Proof. Observe that in the DRW $\mathcal{D}_n$ of Figure 3, each Rabin acceptance pair is actually a co-Büchi condition. Hence, $\mathcal{D}_n$ is the union of DCW's, and is therefore NCW-recognizable. By Theorem 4, an NSW equivalent to $\mathcal{D}_n$ has at least 2^n states, and therefore also an equivalent NCW. ◀

Combining Theorem 6 with known results on the expressive power of the different automata types, we get the following generalization.

▶ **Corollary 7.** *The translations of deterministic Rabin and Muller automata to nondeterministic weak automata, as well as to deterministic and nondeterministic co-Büchi automata involve a state blowup in* $2^{\Theta(n)}$.

Proof. The lower bounds follow from Theorem 6. The upper bounds are given in [2]. ◀

Another consequence of Theorem 6 concerns the translation of deterministic Streett to deterministic Büchi automata. It is known that there is an exponential state blowup in the translation of deterministic Streett to nondeterministic Büchi automata [24]. Yet, the languages used in [24] are not DBW-recognizable, leaving open the translation of DSWs to DBWs. By the duality of DSWs and DRWs and the duality of DBWs and DCWs, we get from Corollary 7 a corresponding answer.

▶ **Corollary 8.** *The translations of deterministic Streett and Muller automata to deterministic Büchi automata involve a state blowup in* $2^{\Theta(n)}$.

Size blowup

We now move from the negative results on the exponential state blowup to positive results on the quadratic size blowup. More precisely, given a DRW with n states and index k that is NCW-recognizable, we construct an equivalent DCW with nk states.

Our construction starts with translating a given DRW with n states and index k to an equivalent NCW, as per the translation of an NRW to an NCW given in [2]. In general, the constructed NCW might have $kn2^n$ states. However, we analyze the special case in which the translated NRW is a DRW, and show that the constructed NCW has up to kn states.

The next step is to determinize the constructed NCW. In general, co-Büchi determinization is done via the breakpoint (Miyano-Hayashi) construction, and might result in an exponential state blowup [17, 3]. Yet, we analyze the constructed NCW to be of a special form, a union of DCWs over the same structure, for which we provide a different determinization construction that introduces no state blowup.

We start with a definition from [2], which provides the central building block in the translations to co-Büchi automata.

▶ **Definition 9** (Augmented subset construction [2]). Let $\mathcal{A} = \langle \Sigma, A, \delta_{\mathcal{A}}, A_0 \rangle$ be an automaton structure. We define its *augmented subset construction* $\mathcal{A}'$ as the product of $\mathcal{A}$ with its subset construction. Formally, $\mathcal{A}' = \langle \Sigma, A', \delta_{\mathcal{A}'}, A'_0 \rangle$, where

- $A' = A \times 2^A$. That is, the states of $\mathcal{A}'$ are all the pairs $\langle a, E \rangle$ where $a \in A$ and $E \subseteq A$.
- For all $\langle a, E \rangle \in A'$ and $\sigma \in \Sigma$, we have $\delta_{\mathcal{A}'}(\langle a, E \rangle, \sigma) = \delta_{\mathcal{A}}(a, \sigma) \times \{\delta_{\mathcal{A}}(E, \sigma)\}$. That is, $\mathcal{A}'$ nondeterministically follows $\mathcal{A}$ on its A-component and deterministically follows the subset construction of $\mathcal{A}$ on its 2^A-component.
- $A'_0 = A_0 \times \{A_0\}$.

We continue with three lemmas from [2], which will serve us in analyzing the constructed NCW.

▶ **Lemma 10** ([2, Lemma 5.3]). *Every NRW and NMW $\mathcal{A}$ with index k is equivalent to the union of k NSWs over the same structure of $\mathcal{A}$.*

▶ **Lemma 11** ([2, Lemma 5.4]). *Consider k NSWs, $\mathcal{S}_1, \ldots, \mathcal{S}_k$, over the same structure. There is an NSW $\mathcal{S}$ over the disjoint union of their structures, such that $L(\mathcal{S}) = \bigcup_{i=1}^{k} L(\mathcal{S}_i)$.*

▶ **Lemma 12** ([2, Proof of Theorem 5.1]). *For every NCW-recognizable NSW $\mathcal{S}$, there is an equivalent NCW $\mathcal{C}$ over the augmented subset construction of $\mathcal{S}$.*

Next, we provide an alternative determinization procedure for an NCW that is a union of DCWs over the same structure. The constructed DCW is generated on the same structure of the given NCW.

The idea is as follows. Consider an NCW $\mathcal{A}$ that is a union of several DCWs over the same structure. A run of $\mathcal{A}$ is accepting if one of the DCWs accepts it. Instead of guessing which DCW will accept it, as the NCW does, a global DCW $\mathcal{D}$ that is equivalent to $\mathcal{A}$ can move between the local DCWs whenever it is in a rejecting state, and remain in the local DCW if it is in an accepting state. Its rejecting states are the union of the rejecting states of the local DCWs. Since the local DCWs share the same structure, there is no harm in moving between them, and if one of them is accepting, the global DCW will eventually remain there forever.

▶ **Lemma 13.** *Consider k DCWs $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$ over the same structure of n states and m transitions. Then there is a DCW $\mathcal{D}$ with nk states and mk transitions that is equivalent to their union, namely $L(\mathcal{D}) = \cup_{i=1}^{k} L(\mathcal{D}_i)$.*

Proof.

Construction. For every $i \in [1..k]$, let $\mathcal{D}_i = \langle \Sigma, Q, q_0, \delta, \alpha_i \rangle$, where the alphabet Σ, the set Q of n states, the initial state q_0, and the transition function δ are common to all the DCWs, while the set $\alpha_i \subseteq Q$ of rejecting states is possibly different in each of them.

We define the DCW $\mathcal{D}' = \langle \Sigma, Q', q'_0, \delta', \alpha' \rangle$, which we claim to recognize $\cup_{i=1}^{k} L(\mathcal{D}_i)$, as follows.

- $Q' = Q \times [1..k]$.
- $q'_0 = \langle q_0, 1 \rangle$.
- For every state $\langle q, i \rangle \in Q'$ and $\sigma \in \Sigma$, the transition function is defined as follows.
  - If $q \in \alpha_i$, then $\delta'(\langle q, i \rangle, \sigma) = \langle \delta(q, \sigma), (i{+}1) \bmod k \rangle$.
  - If $q \notin \alpha_i$, then $\delta'(\langle q, i \rangle, \sigma) = \langle \delta(q, \sigma), i \rangle$.
- For every $q \in Q$ and $i \in [1..k]$, $\langle q, i \rangle \in \alpha'$ iff $q_i \in \alpha_i$;

Correctness. Observe that by the definition of $\mathcal{D}'$, for every position t of a word w, the runs of all $\mathcal{D}_i$'s are at the same state q, while $\mathcal{D}'$ is in a state $\langle q, i \rangle$ for some $i \in [1..k]$. Thus, by the definition of δ', we have:
$w \in L(\mathcal{D}')$ iff
there exists $i \in [1..k]$, such that from some position of w, $\mathcal{D}'$ remains in $(Q_i \setminus \alpha_i) \times \{i\}$ iff
there exists $i \in [1..k]$, such that w is accepted by $\mathcal{D}_i$ iff
$w \in \cup_{i=1}^{k} L(\mathcal{D}_i)$. ◀

We are now in position to provide the upper bound proof. We give it for the translations of both deterministic Rabin and Muller automata.

▶ **Theorem 14.** *For every NCW-recognizable DRW and DMW with n states, m transitions, and index k, there is an equivalent DCW $\mathcal{C}$ with nk states and mk transitions.*

Proof. Consider a DRW or a DMW $\mathcal{A} = \langle \Sigma, A, A_0, \delta, \alpha \rangle$ with n states, m transitions, and index k. By Lemmas 10 and 11, there is an NSW $\mathcal{S}$ equivalent to $\mathcal{A}$ whose structure consists of k copies of the structure of $\mathcal{A}$. That is, $\mathcal{S} = \langle \Sigma, A \times [1..k], A_0 \times [1..k], \delta_{\mathcal{S}}, \alpha_{\mathcal{S}} \rangle$, where for all $a \in A, i \in [1..k]$, and $\sigma \in \Sigma$, we have $\delta_{\mathcal{S}}(\langle a, i \rangle, \sigma) = \langle \delta(a, \sigma), i \rangle$.

Let $\mathcal{C}$ be the NCW equivalent to $\mathcal{S}$, defined over the augmented subset construction of $\mathcal{S}$, as per Lemma 12. Note that $\mathcal{S}$ has nk states, suggesting that an application of the augmented subset construction on it might result in an NCW with up to $nk2^{nk}$ states. Yet, we show below that due to the special structure of $\mathcal{S}$, the resulting NCW $\mathcal{C}$ presents no state blowup, and is defined over a structure that is isomorphic to the structure of $\mathcal{S}$.

Indeed, applying the augmented subset construction on $\mathcal{S}$, we get the product of $\mathcal{S}$ and its subset construction, where the latter has a state for every reachable subset of S. As S consists of k disjoint copies of the same deterministic structure of $\mathcal{A}$, each reachable subset of S is of the form $\{\langle a, 1 \rangle, \langle a, 2 \rangle, \ldots, \langle a, k \rangle\}$, for some $a \in A$. Thus, the subset construction of $\mathcal{S}$ results in a structure that is isomorphic to the structure of $\mathcal{A}$.

Now, as the structure of $\mathcal{C}$ is the product of S and its subset construction, a state of $\mathcal{C}$ is of the form $\langle \langle a, i \rangle, a \rangle$, for some $a \in A$ and $i \in [1..k]$. Hence, the state space of $\mathcal{C}$ is isomorphic to that of $\mathcal{S}$. It remains to see that the transition function $\delta_{\mathcal{C}}$ of $\mathcal{C}$ follows the isomorphism between the states of $\mathcal{S}$ and $\mathcal{C}$. Indeed, for every $a \in A$ and $i \in [1..k]$, $\delta_{\mathcal{C}}(\langle \langle a, i \rangle, a \rangle) = \langle \delta_{\mathcal{S}}(\langle a, i \rangle), \delta(a) \rangle = \langle \langle \delta(a), i \rangle, \delta(a) \rangle$.

Next, we need to determinize the NCW $\mathcal{C}$ into an equivalent DCW. The standard co-Büchi determinization might result in an exponential state blowup [17, 3]. Yet, since the structure of $\mathcal{C}$ is isomorphic to that of $\mathcal{S}$, it follows that $\mathcal{C}$ is the disjoint union of k DCWs over the same structure. Hence, we can determinize it as per Lemma 13, getting a DCW with nk states and mk transitions. ◀

For providing a corresponding lower bound, we look on the dual translation of a DSW to a DBW. Observe that the family of DSWs depicted in Figure 2 are DBW-recognizable. Hence, we get from Theorem 2 the desired bound.

▶ **Theorem 15.** *For every $n \in \mathbb{N}$, there is a DBW-recognizable DSW over a two-letter alphabet with $2n+1$ states, $3n$ transitions, and index n, for which equivalent DBWs have at least $n^2/2$ states.*

Proof. Consider the DSWs $\mathcal{S}_n$ depicted in Figure 2. As explained in the proof of Theorem 2, a run of $\mathcal{S}_n$ is accepting iff it visits all of $\mathcal{S}_n$'s states infinitely often. Hence, $\mathcal{S}_n$ is equivalent to the intersection of n DBWs that are defined over its structure, where each of them has a different single accepting state. As the set of DBW-recognizable languages is closed under intersection, we have that $\mathcal{S}_n$ is DBW-recognizable.

By Theorem 2, every NRW equivalent to $\mathcal{S}_n$ has at least $n^2/2$ states, and therefore also every such DBW. ◀

We conclude with the tight bound.

▶ **Corollary 16.** *The translations of deterministic Rabin automata to deterministic co-Büchi automata and of deterministic Streett automata to deterministic Büchi automata involve a size blowup in $\Theta(n^2)$.*

6 Conclusions

The duality between the Rabin and Streett acceptance conditions, when combined with automata nondeterminism, turns out to result in a duality between the number of states

of an automaton and the length of its acceptance condition (index): We resolve the open problems of the blowup involved in the translations between Rabin and Streett automata, showing that from Rabin to Streett there might be an exponential state blowup, while having no index blowup, whereas from Streett to Rabin there can be only a quadratic state blowup, yet having an exponential index blowup.

Moreover, the state blowup and index blowup are interconnected—The translation from Streett to Rabin can be done not only with a quadratic state blowup and an exponential index blowup, but also with an exponential state blowup and no index blowup; Yet, every algorithm that translates Streett to Rabin must involve either an exponential state blowup or an exponential index blowup.

Following these results, we argue that when studying translations between automata, one should also consider the *size* blowup, where the size of an automaton is the maximum of its elements, namely the alphabet length, the number of states, the number of transitions, and the index. Out of the four elements, the number of states and the index are the dominant ones.

The substantial difference between state blowup and size blowup takes place also in the translations of deterministic Rabin and Streett automata to the weaker types of deterministic co-Büchi and Büchi automata, respectively. We resolve the open problems of the blowup involved in these translations, when possible, showing that the state blowup is exponential and that the size blowup is quadratic.

References

1. U. Boker. On the (in)succinctness of Muller automata. In *CSL*, pages 12:1–12:16, 2017.
2. U. Boker and O. Kupferman. Translating to co-Büchi made tight, unified, and useful. *ACM Trans. Comput. Log.*, 13(4):29:1–29:26, 2012.
3. U. Boker, O. Kupferman, and A. Rosenberg. Alternation removal in Büchi automata. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, volume 6199, pages 76–87, 2010.
4. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.
5. Y. Cai and T. Zhang. Determinization complexities of ω automata. Submitted.
6. Y. Cai and T. Zhang. A tight lower bound for Streett complementation. In *FSTTCS*, pages 339–350, 2011.
7. Y. Cai, T. Zhang, and H. Luo. An improved lower bound for the complementation of Rabin automata. In *LICS*, pages 167–176, 2009.
8. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and Systems Science*, 8:117–141, 1974.
9. T. Colcombet and K. Zdanowski. A tight lower bound for determinization of transition labeled Büchi automata. In *ICALP*, pages 151–162, 2009.
10. Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 60–65. ACM Press, 1982.
11. S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1994.
12. O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. In *2nd Int. Symp. on Automated Technology for Verification and Analysis*, volume 3299 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2004.

13 W. Liu and J. Wang. A tighter analysis of Piterman's Büchi determinization. *Inf. Process. Lett.*, 109(16):941–945, 2009.

14 C. Löding. Optimal bounds for the transformation of omega-automata. In *Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 97–109, 1999.

15 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

16 M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

17 S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

18 A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 1984.

19 D.E. Muller. Infinite sequences and finite machines. In *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical design*, pages 3–16, 1963.

20 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3):5, 2007.

21 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

22 S. Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Institute of Science, 1989.

23 S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, 1992.

24 S. Safra and M.Y. Vardi. On ω-automata and temporal logic. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 127–137, 1989.

25 S. Schewe. Büchi complementation made tight. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

26 S. Schewe and T. Varghese. Determinising parity automata. In *MFCS*, pages 486–498, 2014.

27 R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

28 Qiqi Yan. Lower bounds for complementation of omega-automata via the full automata technique. *Logical Methods in Computer Science*, 4(1), 2008.

How Deterministic are Good-For-Games Automata?*

Udi Boker[†1], **Orna Kupferman**[‡2], **and Michał Skrzypczak**[§3]

1 **Interdisciplinary Center, Herzliya, Israel**
2 **The Hebrew University, Jerusalem, Israel**
3 **University of Warsaw, Poland**

Abstract

In *good for games* (GFG) automata, it is possible to resolve nondeterminism in a way that only depends on the past and still accepts all the words in the language. The motivation for GFG automata comes from their adequacy for games and synthesis, wherein general nondeterminism is inappropriate. We continue the ongoing effort of studying the power of nondeterminism in GFG automata. Initial indications have hinted that every GFG automaton embodies a deterministic one. Today we know that this is not the case, and in fact GFG automata may be exponentially more succinct than deterministic ones.

We focus on the *typeness* question, namely the question of whether a GFG automaton with a certain acceptance condition has an equivalent GFG automaton with a weaker acceptance condition on the same structure. Beyond the theoretical interest in studying typeness, its existence implies efficient translations among different acceptance conditions. This practical issue is of special interest in the context of games, where the Büchi and co-Büchi conditions admit memoryless strategies for both players. Typeness is known to hold for deterministic automata and not to hold for general nondeterministic automata.

We show that GFG automata enjoy the benefits of typeness, similarly to the case of deterministic automata. In particular, when Rabin or Streett GFG automata have equivalent Büchi or co-Büchi GFG automata, respectively, then such equivalent automata can be defined on a substructure of the original automata. Using our typeness results, we further study the place of GFG automata in between deterministic and nondeterministic ones. Specifically, considering automata complementation, we show that GFG automata lean toward nondeterministic ones, admitting an exponential state blow-up in the complementation of a Streett automaton into a Rabin automaton, as opposed to the constant blow-up in the deterministic case.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Finite automata on infinite words, determinism, good-for-games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.18

1 Introduction

Nondeterminism is a prime notion in theoretical computer science. It allows a computing machine to examine, in a concurrent manner, all its possible runs on a certain input. For

* A full version of the paper is available at `http://www.faculty.idc.ac.il/udiboker/files/GFG_Typeness_Full_Version.pdf`.
† This research was supported by the Israel Science Foundation, grant no. 1373/16.
‡ This research has received funding from the European Research Council under the EU's 7-th Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410.
§ Supported by the Polish National Science Centre (decision UMO-2016/21/D/ST6/00491).

automata on finite words, nondeterminism does not increase the expressive power, yet it leads to an exponential succinctness [15]. For automata on infinite words, nondeterminism may increase the expressive power and also leads to an exponential succinctness. For example, nondeterministic Büchi automata are strictly more expressive than their deterministic counterpart [11]. In the automata-theoretic approach to formal verification, we use automata on infinite words in order to model systems and their specifications. In particular, temporal logic formulas are translated to nondeterministic word automata [19]. In some applications, such as model checking, algorithms can proceed on the nondeterministic automaton, whereas in other applications, such as synthesis and control, they cannot. There, the advantages of nondeterminism are lost, and the algorithms involve a complicated determinization construction [16] or acrobatics for circumventing determinization [10]. Essentially, the inherent difficulty of using nondeterminism in synthesis lies in the fact that each guess of the nondeterministic automaton should accommodate all possible futures.

Some nondeterministic automata are, however, good for games: in these automata it is possible to resolve the nondeterminism in a way that only depends on the past while still accepting all the words in the language. This notion, of *good for games* (GFG) automata was first introduced in [4].[1] Formally, a nondeterministic automaton $\mathcal{A}$ over an alphabet Σ is GFG if there is a strategy g that maps each finite word $u \in \Sigma^+$ to the transition to be taken after u is read; and following g results in accepting all the words in the language of $\mathcal{A}$. Note that a state q of $\mathcal{A}$ may be reachable via different words, and g may suggest different transitions from q after different words are read. Still, g depends only on the past, namely on the word read so far. Obviously, there exist GFG automata: deterministic ones, or nondeterministic ones that are *determinizable by pruning* (DetByP); that is, ones that just add transitions on top of a deterministic automaton. In fact, the GFG automata constructed in [4] are DetByP.[2]

Our work continues a series of works that have studied GFG automata: their expressive power, succinctness, and constructions for them, where the key challenge is to understand the power of nondeterminism in GFG automata. Let us first survey the results known so far. In terms of expressive power, it is shown in [8, 14] that GFG automata with an acceptance condition of type γ (e.g., Büchi) are as expressive as deterministic γ automata.[3] Thus, as far as expressiveness is concerned, GFG automata behave like deterministic ones. The picture in terms of succinctness is diverse. For automata on finite words, GFG automata are always DetByP [8, 12]. For automata on infinite words, in particular Büchi and co-Büchi automata[4], GFG automata need not be DetByP [2]. Moreover, the best known determinization construction of GFG Büchi automata is quadratic, whereas determinization of GFG co-Büchi automata has an exponential blow-up lower bound [6]. Thus, in terms of succinctness, GFG automata on infinite words are more succinct (possibly even exponentially) than deterministic ones.

For deterministic automata, where Büchi and co-Büchi automata are less expressive than Rabin and Streett ones, researchers have come up with the notion of an automaton being *type* [5]. Consider a deterministic automaton $\mathcal{A}$ with acceptance condition of type

[1] GFGness is also used in [3] in the framework of cost functions under the name "history-determinism".
[2] As explained in [4], the fact that the GFG automata constructed there are DetByP does not contradict their usefulness in practice, as their transition relation is simpler than the one of the embodied deterministic automaton and it can be defined symbolically.
[3] The results in [8, 14] are given by means of *tree automata for derived languages*, yet, by [2], the results hold also for GFG automata.
[4] See Section 2.1 for the full definition of the various acceptance conditions.

γ and assume that $\mathcal{A}$ recognizes a language that can be recognized by some deterministic automaton with an acceptance condition of type β that is weaker than γ. When deterministic γ automata are β-type, it is guaranteed that a deterministic β-automaton for the language of $\mathcal{A}$ can be defined on top of the structure of $\mathcal{A}$.

For example, deterministic Rabin automata being Büchi-type [5] means that if a deterministic Rabin automaton $\mathcal{A}$ recognizes a language that can be recognized by a deterministic Büchi automaton, then $\mathcal{A}$ has an equivalent deterministic Büchi automaton on the same structure.

Thus, the basic motivation of typeness is to allow simplifications of the acceptance conditions of the considered automata without complicating their structure. Applications of this notion are much wider [5]. In particular, in the context of games, the Büchi and co-Büchi conditions admit memoryless strategies for both players, which is not the case for the Rabin and Streett conditions [18]. Thus, the study of typeness in the context of GFG automata addresses also the question of simplifying the memory requirements of the players. In addition, as we elaborate in Section 7, it leads to new and non-trivial bounds on the blow-up of transformations between GFG automata and their complementation.

Recall that deterministic Rabin automata are Büchi-type. Dually, deterministic Streett automata are co-Büchi-type. Typeness can be defined also with respect to nondeterministic automata, yet it crucially depends on the fact that the automaton is deterministic. Indeed, nondeterministic Rabin are not Büchi-type. Even with the co-Büchi acceptance condition, where nondeterministic co-Büchi automata recognize only a subset of the ω-regular languages, nondeterministic Streett automata are not co-Büchi-type [7].

We first show that typeness is strongly related with determinism even when slightly relaxing the typeness notion to require the existence of an equivalent automaton on a substructure of the original automaton, instead of on the exact original structure, and even when we restrict attention to an *unambiguous* automaton, namely one that has a single accepting run on each word in its language. We describe an unambiguous parity automaton $\mathcal{A}$, such that its language is recognized by a deterministic Büchi automaton, yet it is impossible to define a Büchi acceptance condition on top of a substructure of $\mathcal{A}$. We also point to a dual result in [7], with respect to the co-Büchi condition, and observe that it applies also to the relaxed typeness notion.

We then show that for GFG automata, typeness, in its relaxed form, does hold. Notice that once considering GFG automata with no redundant transitions, which we call *tight*, the two typeness notions coincide. Obviously, all GFG automata can be tightened by removal of redundant transitions (Lemma 4). In particular, we show that the typeness picture in GFG automata coincides with the one in deterministic automata: Rabin GFG automata are Büchi type, Streett GFG automata are co-Büchi type, and all GFG automata are type with respect to the weak acceptance condition. Unlike the deterministic case, however, the Rabin case is not a simple dualization of the Streett case; it is much harder to prove and it requires a stronger notion of tightness.

We continue with using our typeness results for further studying the place of GFG automata in between deterministic and nondeterministic ones. We start with showing that all GFG automata that recognize languages that can be defined by deterministic weak automata are DetByP. This generalizes similar results about safe and co-safe languages [7]. We then show that all unambiguous GFG automata are also DetByP. Considering complementation, GFG automata lie in between the deterministic and nondeterministic settings—the complementation of a Büchi automaton into a co-Büchi automaton is polynomial, as is the case with deterministic automata, while the complementation of a co-Büchi automaton into a Büchi

automaton as well as the complementation of a Streett automaton into a Rabin automaton is exponential, as opposed to the constant blow-up in the deterministic case. We conclude with proving a doubly-exponential lower bound for the translation of LTL into GFG automata, as is the case with deterministic automata.

The paper is structured as follows. In Section 2 we provide the relevant notions about languages and GFG automata. Section 3 contains examples showing that typeness does not hold for the case of unambiguous automata. The next three sections, Sections 4, 5, and 6, provide the main positive results of this work: co-Büchi typeness for GFG-Streett; Büchi typeness for GFG-Rabin; and weak typeness for GFG-Büchi and GFG-co-Büchi, respectively. Finally, in Section 7 we continue to study the power of nondeterminism in GFG automata, looking into automata complementation and translations of LTL formulas to GFG automata. Due to lack of space, some full proofs are missing, and can be found in the full version, in the authors' URLs.

2 Preliminaries

2.1 Automata

An automaton on infinite words is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Σ is an input alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta \colon Q \times \Sigma \to 2^Q$ is a transition function that maps a state and a letter to a set of possible successors, and α is an acceptance condition. The first four elements, namely $\langle \Sigma, Q, \delta, Q_0 \rangle$, are the automaton's *structure*. We consider here the *Büchi*, *co-Büchi*, *parity*, *Rabin*, and **Streett** acceptance conditions. (The *weak* condition is defined in Section 6.) In Büchi, and co-Büchi conditions, $\alpha \subseteq Q$ is a set of states. In a parity condition, $\alpha \colon Q \to \{0, \ldots, k\}$ is a function mapping each state to its priority. In a Rabin and Streett conditions, $\alpha \subseteq 2^{2^Q \times 2^Q}$ is a set of pairs of sets of states. The *index* of a Rabin or Streett condition is the number of pairs in it. For a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by changing the set of initial states to $\{q\}$. A *transition* of $\mathcal{A}$ is a triple $\langle q, a, q' \rangle$ such that $q' \in \delta(q, a)$. We extend δ to sets of states and to finite words in the expected way. Thus, for a set $S \subseteq Q$, a letter $a \in \Sigma$, and a finite word $u \in \Sigma^*$, we have that $\delta(S, \epsilon) = S$, $\delta(S, a) = \bigcup_{q \in S} \delta(q, a)$, and $\delta(S, u \cdot a) = \delta(\delta(S, u), a)$. Then, we denote by $\mathcal{A}(u)$ the set of states that $\mathcal{A}$ may reach when reading u. Thus, $\mathcal{A}(u) = \delta(Q_0, u)$.

Since the set of initial states need not be a singleton and the transition function may specify several successors for each state and letter, the automaton $\mathcal{A}$ may be *nondeterministic*. If $|Q_0| = 1$ and $|\delta(q, a)| \leq 1$ for every $q \in Q$ and $a \in \Sigma$, then $\mathcal{A}$ is *deterministic*.

Given an input word $w = a_1 \cdot a_2 \cdots$ in Σ^ω, a *run* of $\mathcal{A}$ on w is an infinite sequence $r = r_0, r_1, r_2, \ldots \in Q^\omega$ such that $r_0 \in Q_0$ and for every $i \geq 0$, we have $r_{i+1} \in \delta(r_i, a_{i+1})$; i.e., the run starts in the initial state and obeys the transition function. For a run r, let $\inf(r)$ denote the set of states that r visits infinitely often. That is, $\inf(r) = \{q \in Q \mid \text{for infinitely many } i \geq 0, \text{ we have } r_i = q\}$.

A set of states S satisfies an acceptance condition α (or *is accepting*) iff

- $S \cap \alpha \neq \emptyset$, for a Büchi condition.
- $S \cap \alpha = \emptyset$, for a co-Büchi condition.
- $\min_{q \in \inf(r)}\{\alpha(q)\}$ is even, for a parity condition.
- There exists $\langle E, F \rangle \in \alpha$, such that $S \cap E = \emptyset$ and $S \cap F \neq \emptyset$ for a Rabin condition.
- For all $\langle E, F \rangle \in \alpha$, we have $S \cap E = \emptyset$ or $S \cap F \neq \emptyset$ for a Streett condition.

Notice that Büchi and co-Büchi are dual, and so are Rabin and Streett. Also note that the Büchi and co-Büchi conditions are special cases of parity, which is a special case of Rabin and Streett. In the latter conditions, we refer to the sets E and F as the "bad" and "good"

sets, respectively. Finally, note that a Rabin pair may have an empty E component, while an empty F component makes the pair redundant (and dually for Streett).

A run r is accepting if $\inf(r)$ satisfies α. An automaton $\mathcal{A}$ accepts an input word w iff there exists an accepting run of $\mathcal{A}$ on w. The *language* of $\mathcal{A}$, denoted by $\mathrm{L}(\mathcal{A})$, is the set of all words in Σ^ω that $\mathcal{A}$ accepts. A nondeterministic automaton $\mathcal{A}$ is *unambiguous* if for every word $w \in \mathrm{L}(\mathcal{A})$, there is a single accepting run of $\mathcal{A}$ on w. Thus, while $\mathcal{A}$ is nondeterministic and may have many runs on each input word, it has only a single accepting run on words in its language.

We denote the different automata types by three-letter acronyms in the set $\{\mathrm{D},\mathrm{N}\} \times \{\mathrm{B},\mathrm{C},\mathrm{P},\mathrm{R},\mathrm{S}\} \times \{\mathrm{W}\}$. The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second for the acceptance-condition type (Büchi, co-Büchi, parity, Rabin, or Streett); and the third indicates that we consider automata on words. For Rabin and Streett automata, we sometimes also indicate the index of the automaton. In this way, for example, NBW are nondeterministic Büchi word automata, and DRW[1] are deterministic Rabin automata with index 1.

For two automata $\mathcal{A}$ and $\mathcal{A}'$, we say that $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{A}')$. For an automaton type β (e.g., DBW) and an automaton $\mathcal{A}$, we say that $\mathcal{A}$ is β-realizable if there is a β-automaton equivalent to $\mathcal{A}$.

Let $\mathcal{A} = \langle \mathcal{A}, Q, Q_0, \delta, \alpha \rangle$ be an automaton. For an acceptance-condition class γ (e.g., Büchi), we say that $\mathcal{A}$ is *γ-type* if $\mathcal{A}$ has an equivalent γ automaton with the same structure as $\mathcal{A}$ [5]. That is, there is an automaton $\mathcal{A}' = \langle \Sigma, Q, Q_0, \delta, \alpha' \rangle$ such that α' is an acceptance condition of the class γ and $\mathrm{L}(\mathcal{A}') = \mathrm{L}(\mathcal{A})$.

2.2 Good-For-Games Automata

An automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$ is *good for games* (GFG, for short) if there is a strategy $g\colon \Sigma^* \to Q$, such that for every word $w = a_1 \cdot a_2 \cdots \in \Sigma^\omega$, the sequence $g(w) = g(\epsilon)$, $g(a_1)$, $g(a_1 \cdot a_2)$,... is a run of $\mathcal{A}$ on w, and whenever $w \in \mathrm{L}(\mathcal{A})$, then $g(w)$ is accepting. We then say that g *witnesses* $\mathcal{A}$'s GFGness.

It is known [2] that if $\mathcal{A}$ is GFG, then its GFGness can be witnessed by a finite-state strategy, thus one in which for every state $q \in Q$, the set of words $g^{-1}(q)$ is regular. Finite-state strategies can be modeled by *transducers*. Given sets I and O of input and output letters, an *(I/O)-transducer* is a tuple $\mathcal{T} = \langle I, O, M, m_0, \rho, \tau \rangle$, where M is a finite set of states, to which we refer as *memories*, $m_0 \in M$ is an *initial memory*, $\rho\colon M \times I \to M$ is a deterministic transition function, to which we refer as the *memory update function*, and $\tau\colon M \to O$ is an output function that assigns a letter in O to each memory. The transducer $\mathcal{T}$ generates a strategy $g_{\mathcal{T}}\colon I^* \to O$, obtained by following ρ and τ in the expected way: we first extend ρ to words in I^* by setting $\rho(\epsilon) = m_0$ and $\rho(u \cdot a) = \rho(\rho(u), a)$, and then define $g_{\mathcal{T}}(u) = \tau(\rho(u))$.

Consider a GFG automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, and let $g = \langle \Sigma, Q, M, m_0, \rho, \tau \rangle$ be a finite-state (Σ/Q)-transducer that generates a strategy $g\colon \Sigma^* \to Q$ that witnesses $\mathcal{A}$'s GFGness (we abuse notations and use g to denote both the transducer and the strategy it generates). Consider a state $q \in Q$. When $\tau(m) = q$, we say that m is a *memory of* q. We denote by $\mathcal{A}_g$ the (deterministic) automaton that models the operation of $\mathcal{A}$ when it follows g. Thus, $\mathcal{A}_g = \langle \Sigma, M, m_0, \rho, \alpha_g \rangle$, where the acceptance condition α_g is obtained from α by replacing each set $F \subseteq Q$ that apndinpears in α (e.g. accepting states, rejecting states, set in a Rabin or Streett pair, etc) by the set $F_g = \{m \mid \tau(m) \in F\}$. Thus, $F_g \subseteq M$ contains the memories of F's states. For a state q of $\mathcal{A}$, a path π of $\mathcal{A}_g$ is *q-exclusive accepting* if π is accepting, and $\inf(\pi) \setminus \{m \mid m \text{ is a memory of } q\}$ is not accepting.

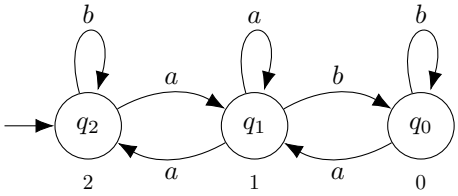


Figure 1 A weakly tight GFG-NPW $\mathcal{A}_0$. The numbers below the states describe their priorities.

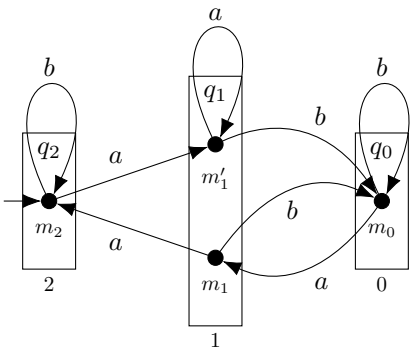


Figure 2 A strategy witnessing the GFG-ness of the automaton $\mathcal{A}_0$, depicted in Figure 1.

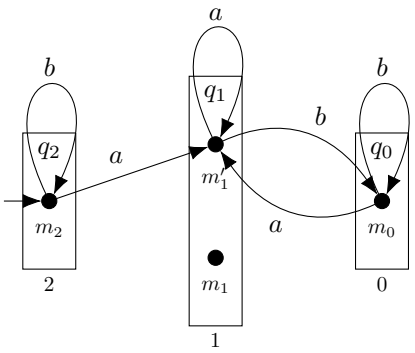


Figure 3 A strategy witnessing the tightness of a sub-automaton of $\mathcal{A}_0$.

▶ **Example 1.** Consider the NPW $\mathcal{A}_0$ appearing in Figure 1. We claim that $\mathcal{A}_0$ is a GFG-NPW that recognizes the language $L_0 = \{w \in \{a,b\}^\omega \mid \text{there are infinitely many } b\text{'s in } w\}$.

Proof sketch. If a word w contains only finitely many b's then $\mathcal{A}_0$ rejects w, as in all the runs of $\mathcal{A}_0$ on w, the lowest priority appearing infinitely often is 1. Therefore, $\mathrm{L}(\mathcal{A}_0) \subseteq L_0$.

We turn to describe a strategy $g\colon \{a,b\}^* \to Q$ with which $\mathcal{A}_0$ accepts all words in L_0. The only nondeterminism in $\mathcal{A}_0$ is when reading the letter a in the state q_1. Thus, we have to describe g only for words that reach q_1 and continue with an a. In that case, the strategy g moves to the state q_2, if the previous state is q_0, and to the state q_1, otherwise. Figure 2 describes a (Σ/Q)-transducer that generates g. The rectangles denote the states of $\mathcal{A}_0$, while the dots are their g-memories. The numbers below the rectangles describe the priorities of the respective states of $\mathcal{A}_0$. It is easy to check that if $w \in L_0$ (i.e. w contains infinitely many b's) then $\mathcal{A}_{0g}$ accepts w. ◀

The following lemma generalizes known residual properties of GFG automata (c.f., [6]).

▶ **Lemma 2.** *Consider a GFG automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$ and let $g = \langle \Sigma, Q, M, m_0, \rho, \tau\rangle$ be a strategy witnessing its GFGness.*

(1) *For every state $q \in Q$ and memory $m \in M$ of q that is reachable in $\mathcal{A}_g$, we have that $\mathrm{L}(\mathcal{A}_g^m) = \mathrm{L}(\mathcal{A}^q)$.*

(2) *For every memories $m, m' \in M$ that are reachable in $\mathcal{A}_g$ with $\tau(m) = \tau(m')$, we have that $\mathrm{L}(\mathcal{A}_g^m) = \mathrm{L}(\mathcal{A}_g^{m'})$.*

A finite *path* $\pi = q_0, \ldots, q_k$ in $\mathcal{A}$ is a sequence of states such that for $i = 0, \ldots, k-1$ we have $q_{i+1} \in \delta(q_i, a_i)$ for some $a_i \in \Sigma$. A path is a *cycle* if $q_0 = q_k$. Each path π induces a set $\mathit{states}(\pi) = \{q_0, \ldots, q_k\}$ of states in Q. A set S of finite paths then induces the set $\mathit{states}(S) = \bigcup_{\pi \in S} \mathit{states}(\pi)$. For a set P of finite paths, a *combination of paths from P* is a set $\mathit{states}(S)$ for some nonempty $S \subseteq P$.

Consider a strategy $g = \langle \Sigma, Q, M, m_0, \rho, \tau \rangle$. We say that a transition $\langle q, a, q' \rangle$ of $\mathcal{A}$ is *used by* g if there is a word $u \in \Sigma^*$ and a letter $a \in \Sigma$ such that $q = g(u)$ and $q' = g(u \cdot a)$. Consider two memories $m \neq m' \in M$ with $\tau(m) = \tau(m')$. Let $P_{m' \to m}$ be the set of paths of $\mathcal{A}_g$ from m' to m. We say that m is *replaceable by* m' if $P_{m' \to m}$ is empty or all combinations of paths from $P_{m' \to m}$ are accepting.

We say that $\mathcal{A}$ is *tight with respect to* g if all the transitions of $\mathcal{A}$ are used in g, and for all memories $m \neq m' \in M$ with $\tau(m) = \tau(m')$, we have that m is not replaceable by m'. Intuitively, the latter condition implies that both m and m' are required in g, as an attempt to merge them strictly reduces the language of $\mathcal{A}_g$. When only the first condition holds, namely when all the transitions of $\mathcal{A}$ are used in g, we say that $\mathcal{A}$ is *weakly tight* with respect to g. When a Rabin automaton $\mathcal{A}$ is *tight with respect to* g, and in addition for every state q that appears in some good set of $\mathcal{A}$'s acceptance condition, there is a q-exclusive accepting cycle in $\mathcal{A}_g$, we say that $\mathcal{A}$ is *strongly tight* with respect to g. Then, $\mathcal{A}$ is (weakly, strongly) *tight* if it is (weakly, strongly) tight with respect to some strategy.

▶ **Example 3.** The GFG-NPW $\mathcal{A}_0$ from Example 1 is weakly tight and is not tight with respect to the strategy g. Indeed, while all the transitions in $\mathcal{A}_0$ are used in g, the memory m_1 is replaceable by m_1', as all combinations of paths from m_1' to m_1 are accepting. ◀

The following lemma formalizes the intuition that every GFG automaton can indeed be restricted to its tight part, by removing redundant transitions and memories. Further, every tight Rabin GFG automaton has an equivalent strongly tight automaton over the same structure.

▶ **Lemma 4.** *For every GFG automaton $\mathcal{A}$ there exists an equivalent tight GFG automaton $\mathcal{A}'$. Moreover, $\mathcal{A}'$ is defined on a substructure of $\mathcal{A}$.*

▶ **Lemma 5.** *For every tight Rabin GFG automaton, there exists an equivalent strongly tight Rabin GFG automaton over the same structure.*

▶ **Example 6.** In Figure 3 we describe a strategy g' that witnesses the tightness of a GFG-NPW on a substructure of the GFG-NPW $\mathcal{A}$ from Example 1. The strategy g' is obtained from g by following the procedure described in the proof of Lemma 4: all the transitions to m_1 are redirected to m_1'. This causes the transition (q_1, a, q_2) that was used by the memory m_1 not to be used, and it is removed. ◀

A special case of GFG automata are those who are *determinizable by pruning* (or shortly DetByP) — there exists a state $q_0 \in Q_0$ and a function $\delta' \colon Q \times \Sigma \to Q$ that for every state q and letter a satisfies $\delta'(q, a) \in \delta(q, a)$ such that $\mathcal{A}' = \langle \Sigma, Q, q_0, \delta', \alpha \rangle$ is a deterministic automaton recognizing the language $\mathrm{L}(\mathcal{A})$.

3 Typeness Does Not Hold for Unambiguous Automata

As noted in [7], it is easy to see that typeness does not hold for nondeterministic automata: there exists an NRW that recognizes an NBW-realizable language, yet does not have an equivalent NBW on the same structure. Indeed, since all ω-regular languages are NBW-realizable, typeness in the nondeterministic setting would imply a translation of all NRWs to NBWs on the same structure, and we know that such a translation may involve a blow-up linear in the index of the NRW [17]. Even for Streett and co-Büchi automata, where the restriction to NCW-realizable languages amounts to a restriction to DCW-realizable languages, typeness does not hold.

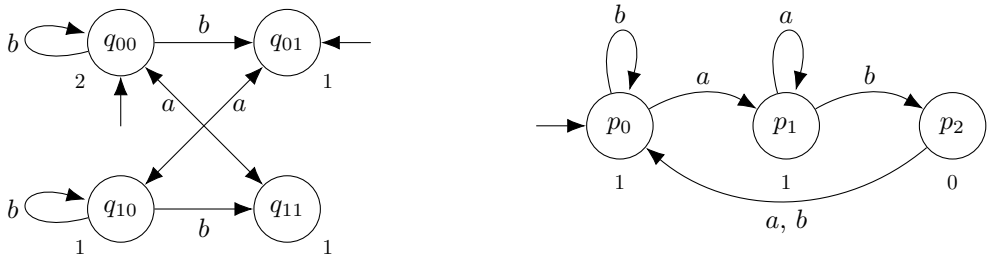


Figure 4 $\mathcal{A}_1$: An unambiguous NPW that is DBW-realizable yet is not Büchi-type.

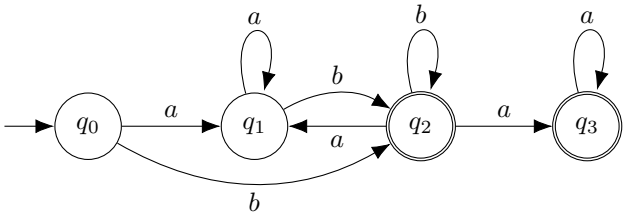


Figure 5 $\mathcal{A}_2$: An unambiguous NBW that is DCW-realizable yet is not co-Büchi-type.

In this section we strengthen the relation between typeness and determinism and show that typeness does not hold for nondeterministic automata even when they recognize a DBW-realizable language and, moreover, when they are unambiguous. Also, we prove the non-typeness results for NPWs, thus they apply to both Rabin and Streett automata.

▶ **Example 7.** Unambiguous NPWs are not Büchi-type with respect to DBW-realizable languages: The automaton $\mathcal{A}_1$ depicted in Figure 4 is unambiguous and recognizes a DBW-realizable language, yet $\mathcal{A}_1$ is not Büchi-type. Moreover, we cannot prune transitions from $\mathcal{A}_1$ and obtain an equivalent Büchi-type NPW.

The dual case of unambiguous NPWs that are not co-Büchi-type with respect to DCW-realizable languages follows from the results of [7], and we give it here for completeness, adding the observation that the automaton described there cannot be pruned to an equivalent co-Büchi-type NPW.

▶ **Example 8.** [7] Unambiguous NPWs (and even NBWs) are not co-Büchi-type with respect to DCW-realizable languages: The NBW $\mathcal{A}_2$ depicted in Figure 5 is unambiguous, and recognizes a DCW-realizable language, yet $\mathcal{A}_2$ is not co-Büchi-type. Moreover, we cannot prune transitions from $\mathcal{A}_2$ for obtaining an equivalent co-Büchi-type NPW.

We conclude this section with the following rather simple proposition, showing that automata that are both unambiguous and GFG are essentially deterministic. Essentially, it follows from the fact that by restricting an unambiguous GFG automaton $\mathcal{A}$ to reachable and nonempty states, we obtain, by pruning, a deterministic automaton, which is clearly equivalent to $\mathcal{A}$.

▶ **Proposition 9.** *Unambiguous GFG automata are DetByP.*

4 Co-Büchi Typeness for GFG-NSWs

In this section we study typeness for GFG-NSWs and show that, as is the case with deterministic automata, tight GFG-NSWs are co-Büchi-type. On a more technical level, the

proof of Theorem 10 only requires the GFG automata to be weakly tight (rather than fully tight), implying that Theorem 10 can be strengthened in accordance. This fact is considered in Section 5, where the typeness of GFG-NRWs is shown to require full tightness.

▶ **Theorem 10.** *Tight GFG-NSWs are co-Büchi-type: Every tight GFG-NSW that recognizes a GFG-NCW-realizable language has an equivalent GFG-NCW on the same structure.*

Proof sketch. Given a GFG-NSW $\mathcal{A}$ and a strategy g that witnesses its GFGness, we change $\mathcal{A}$ into an NCW $\mathcal{A}'$ by defining the co-Büchi acceptance condition α' to include the states all of whose g-memories only belong to rejecting cycles in $\mathcal{A}_g$.

We then prove that $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{A}')$ and that $\mathcal{A}'$ is indeed GFG. (Furthermore, we show that the original strategy g also witnesses the GFGness of $\mathcal{A}'$.) The proof goes by induction, iterating over all states q of $\mathcal{A}$, and gradually changing the acceptance condition until it becomes a co-Büchi condition. We show that at each step of the induction, the resulting automaton is still a GFG-NSW that recognizes the original language.

We start the induction with the original GFG-NSW $\mathcal{A}$, and add to its acceptance condition a new empty Streett pair, namely $(\emptyset, \emptyset)$. Along the induction, handling a state q, we remove q from all the original "bad" sets (namely the left components) of the acceptance condition, and in the case that $q \in \alpha'$, we add it to the bad set of the new acceptance pair. Observe that at the end of the induction, the acceptance condition consists of a single Streett pair, in which α' is the bad set and $\emptyset$ is the good set. Thus, it is in fact the NCW $\mathcal{A}'$.

The tricky part of the induction step is when the considered state q does not belong to α'. In this case, removing it from the bad sets might enlarge the language of the automaton. To prove that the language is not altered, we take advantage of the fact that there exists a deterministic co-Büchi automaton $\mathcal{D}$ equivalent to $\mathcal{A}$, and provide a pumping scheme that proceeds along the cycles of $\mathcal{A}$ and $\mathcal{D}$.

Analyzing these cycles, we use the (weak) tightness of $\mathcal{A}$, in order to use Lemma 2—the pumped cycles might go through states of $\mathcal{A}$ that were not originally visited. Yet, due to Lemma 2, we may link the residual languages of the originally visited states with that of the newly visited ones. ◀

The following example shows that the weak tightness requirement cannot be omitted, even when the GFG-NSW is actually a GFG-NBW.

▶ **Example 11.** The automaton $\mathcal{A}_3$ depicted in Figure 7 is GFG-NBW and recognizes a GFG-NCW-realizable language, yet $\mathcal{A}_3$ has no equivalent NCW on the same structure.

5 Büchi Typeness for GFG-NRWs

Studying typeness for deterministic automata, one can use the dualities between the Büchi and co-Büchi, as well as the Rabin and Streett conditions, in order to relate the Büchi-typeness of DRWs with the co-Büchi typeness of DSWs. In the nondeterministic setting, we cannot apply duality considerations, as by dualizing a nondeterministic automaton, we obtain a universal one. As we shall see in this section, our inability to use dualization considerations is not only technical. There is an inherent difference between the co-Büchi typeness of GFG-NSWs studied in Section 4, and the Büchi typeness of GFG-NRWs, which we study here. We first show that while the proof of Theorem 10 only requires weak tightness, Büchi typeness requires full tightness.

The following example shows that tightness is necessary already for GFG-NCW that are GFG-NBW-realizable.

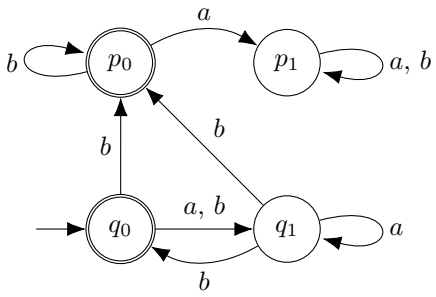


Figure 6 $\mathcal{A}_4$: A weakly tight GFG-NCW that is GFG-NBW-realizable yet is not Büchi-type.

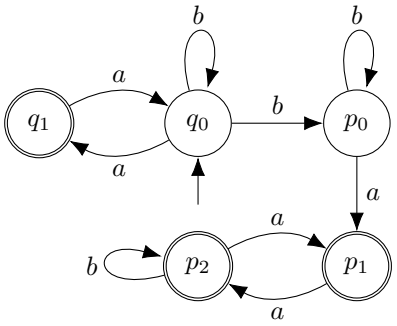


Figure 7 $\mathcal{A}_3$: A GFG-NBW that is GFG-NCW-realizable yet is not co-Büchi-type.

▶ **Example 12.** The automaton $\mathcal{A}_4$ depicted in Figure 6 is a weakly tight GFG-NCW that recognizes a GFG-NBW-realizable language, yet $\mathcal{A}_4$ has no equivalent GFG-NBW on the same structure.

We now proceed to our main positive result, obtaining the typeness of GFG-NRWs.

▶ **Theorem 13.** *Tight GFG-NRWs are Büchi-type: Every tight GFG-NRW that recognizes a GFG-NBW-realizable language has an equivalent GFG-NBW on the same structure.*

Proof sketch. Consider a tight GFG-NRW $\mathcal{A}$ that recognizes a GFG-NBW-realizable language. Let g be a strategy that witnesses $\mathcal{A}$'s GFGness and with respect to which $\mathcal{A}$ is tight. By Lemma 5, we have a GFG Rabin automaton $\mathcal{A}'$ over the structure of $\mathcal{A}$ that is strongly tight with respect to g. We define an NBW $\mathcal{B}$ on top of $\mathcal{A}$'s structure, setting its accepting states to be all the states that appear in "good" sets of $\mathcal{A}'$ (namely in the right components of the Rabin accepting pairs).

Clearly, $\mathrm{L}(\mathcal{A}') \subseteq \mathrm{L}(\mathcal{B})$, as $\mathcal{B}$'s condition only requires the "good" part of $\mathcal{A}'$'s condition, without requiring to visit finitely often in a corresponding "bad" set. We should thus prove that $\mathrm{L}(\mathcal{B}) \subseteq \mathrm{L}(\mathcal{A}')$ and that $\mathcal{B}$ is GFG. Once proving the language equivalence, $\mathcal{B}$'s GFGness is straight forward, as the strategy g witnesses it. The language equivalence, however, is not at all straightforward.

In order to prove that $\mathrm{L}(\mathcal{B}) \subseteq \mathrm{L}(\mathcal{A}')$, we analyze the cycles of $\mathcal{A}'$ and of $\mathcal{A}'_g$. The proof goes along the following steps:

- We start with showing that a memory of a state q cannot belong to both a q-exclusive accepting cycle and a rejecting cycle. In order to prove that, we take advantage of the fact that there is a DBW $\mathcal{D}$ equivalent to $\mathcal{A}'$, and provide a pumping argument concerning the cycles of $\mathcal{A}'$ and $\mathcal{D}$.
- By the above and the strong tightness of $\mathcal{A}'$, we show that every state q of $\mathcal{A}'$ that appears in some good set has a single memory, all the cycles of $\mathcal{A}'_g$ that include q are accepting, and at least one of them is q-exclusive. Observe that this implies the language containment $\mathrm{L}(\mathcal{A}'_g) \subseteq \mathrm{L}(\mathcal{B})$. Yet, it does not guarantee that $\mathrm{L}(\mathcal{A}') \subseteq \mathrm{L}(\mathcal{B})$, as there might be some rejecting cycle of $\mathcal{A}'$ that $\mathcal{A}'_g$ does not use, which becomes accepting by our changes to the acceptance condition.
- For precluding the aforementioned concern and concluding our proof, we show that every state q of $\mathcal{A}'$ that appears in some good set does not belong to a rejecting cycle. We

prove it by applying a maximality argument on paths that the strategy g exhausts along a hypothetical rejecting cycle of $\mathcal{A}'$ that includes q. ◀

The following result follows directly from Lemma 4, Theorem 13, and the determinization procedure for Büchi GFG automata from [6].

▶ **Corollary 14.** *Every GFG-NRW with n states that recognizes a DBW-realizable language has an equivalent DBW with at most $O(n^2)$ states.*

6 Weak Typeness for GFG Automata

A Büchi automaton $\mathcal{A}$ is *weak* [13] if for each strongly connected component C of $\mathcal{A}$, either $C \subseteq \alpha$ (in which case we say that C is an *accepting component*) or $C \cap \alpha = \emptyset$ (in which case we say that C is a *rejecting component*). Note that a weak automaton can be viewed as both a Büchi and a co-Büchi automaton, as a run of $\mathcal{A}$ visits α infinitely often iff it gets trapped in an accepting component iff it visits states in $Q \setminus \alpha$ only finitely often. We use NWW and DWW to denote nondeterministic and deterministic weak word automata, respectively.

We show in this section that all GFG automata are type with respect to the weak acceptance condition. We provide the theorem with respect to GFG-NCWs, from which we can easily deduce it, by our previous typeness results, also for the other types.

▶ **Theorem 15.** *Tight GFG-NCWs are weak-type: every tight GFG-NCW that recognizes a GFG-NWW-realizable language has an equivalent GFG-NWW on the same structure.*

Proof sketch. Consider a tight GFG-NCW $\mathcal{A}$ that recognizes a GFG-NWW-realizable language. Let S be the set of rejecting states of $\mathcal{A}$ and let g be a strategy witnessing $\mathcal{A}$'s tight GFGness. Let S' be the union of S and all the states q of $\mathcal{A}$ for which no g-memory m has an accepting cycle in $\mathcal{A}_g$. Let $\mathcal{A}'$ be the automaton $\mathcal{A}$ with the co-Büchi condition given by S'. Analyzing the cycles of $\mathcal{A}'$, we show that it is a GFG-NWW equivalent to $\mathcal{A}$. ◀

Consider now a GFG-NSW $\mathcal{A}$ that is GFG-NWW-realizable. Notice that it is obviously also GFG-NBW-realizable. Hence, by Theorem 10, there is a GFG-NCW on $\mathcal{A}$'s structure, and by Theorem 15 also a GFG-NWW. The cases of a GFG-NPW and a GFG-NBW obviously follow, since they are special cases of GFG-NSWs. As for a GFG-NRW $\mathcal{A}$ that is GFG-NWW-realizable, notice that it is obviously also GFG-NBW-realizable. Hence, by Theorem 13, there is a GFG-NBW on $\mathcal{A}$'s structure, and by Theorem 15 also a GFG-NWW.

▶ **Corollary 16.** *Tight GFG-NSWs and GFG-NRWs are weak-type: every tight GFG-NSW and GFG-NRW that recognizes a GFG-NWW-realizable language has an equivalent GFG-NWW on the same structure.*

Next, we show that GFG-NWWs are DetByP, generalizing a folklore result about safe and co-safe GFG automata.

▶ **Theorem 17.** *GFG-NWWs are DetByP.*

Proof sketch. Consider a GFG-NWW $\mathcal{A}$ with accepting set α, whose GFGness is witnessed by a strategy g. By arguments analogous to those made in the proof of Theorem 13, a state $q \in \alpha$ has only one g-memory, and is therefore already deterministic.

The interesting part concerns a state $q \notin \alpha$ that has at least two g-memories m and m'. Let g' be the strategy obtained by removing m' from g and redirecting transitions to m' into m. We claim that $\mathrm{L}(\mathcal{A}_g) = \mathrm{L}(\mathcal{A}_{g'})$, from which it follows by induction that the number of memories of each state of $\mathcal{A}$ can be reduced to 1. We prove the claim by induction on the strongly connected components of $\mathcal{A}$ and by inductively applying Lemma 2. ◀

By combining the above results, we obtain the following corollary.

▶ **Corollary 18.** *Every GFG-NSW and GFG-NRW that recognizes a GFG-NWW-realizable language is DetByP.*

7 Consequences

GFG automata provide an interesting formalism in between deterministic and nondeterministic automata. Their translation to deterministic automata is immediate for the weak condition (Theorem 17), polynomial for the Büchi condition [6], and exponential for the co-Büchi, parity, Rabin, and Streett conditions [6]. They have the same typeness behavior as deterministic automata, summarized in Table 1. The positive results of Table 1 follow from our theorems in Sections 4, 5, and 6. The negative results follow from corresponding counterexamples with deterministic automata [5, 7]. Considering the complementation of GFG automata, they lie in between the deterministic and nondeterministic settings, as shown in Table 2. As for the translation of LTL formulas to GFG automata, it is doubly exponential, like the translation to deterministic automata (Corollary 21 below).

Complementation

In the deterministic setting, Rabin and Streett automata are dual: complementing a DRW into a DSW, and vice versa, is simply done by switching between the two acceptance conditions on top of the same structure. This is not the case with GFG automata. We show below that complementing a GFG-NSW, and even a GFG-NCW, into a GFG-NRW involves an exponential state blow-up. Essentially, it follows from the Büchi-typeness of GFG-NRWs (Theorem 13) and the fact that while determinization of GFG-NBWs involves only a quadratic blow-up, determinization of GFG-NCWs involves an exponential one [6].

▶ **Corollary 19.** *The complementation of a GFG-NCW into a GFG-NRW involves a $2^{\Omega(n)}$ state blow-up.*

Using our typeness results, we get an almost complete picture on complementation of GFG automata.

▶ **Theorem 20.** *The state blow-up involved in the complementation of GFG automata is as summarized in Table 2.*

Proof.

- From weak and Büchi. A GFG-NBW $\mathcal{A}$ with n states has an equivalent DBW $\mathcal{D}$ with up to n^2 states [6], on which structure there is a DCW $\overline{\mathcal{D}}$ for the complement language. Notice that $\overline{\mathcal{D}}$ is also a GFG-NCW, GFG-NPW, GFG-NRW, and GFG-NSW. Now, if there is a GFG-NBW equivalent to $\overline{\mathcal{D}}$, then $\overline{\mathcal{D}}$ is DWW-recognizable, and, by Theorem 15, there is a GFG-NWW on a substructure of $\overline{\mathcal{D}}$.
- From co-Büchi. By Corollary 19, we have the exponential state blow-up in the complementation to GFG-NPW and GFG-NRW automata. Since the complement of a co-Büchi-recognizable language is DBW-recognizable, we get an exponential state blow-up also to GFG-NBW.
- To weak and co-Büchi. Consider a GFG-NCW, GFG-NPW, or GFG-NRW $\mathcal{A}$ with n states that can be complemented into a GFG-NCW $\mathcal{C}$. Then the language of $\mathcal{A}$ is GFG-NBW recognizable. Thus, by Theorem 13, there is a GFG-NBW equivalent to $\mathcal{A}$ with up to n states. Hence, by case (1), there is a GFG-NCW for the complement of $\mathcal{A}$ with up to n^2 states.

Table 1 Typeness in translations between GFG automata. (Y=Yes; N=No.)

Type To / From	W	B	C	P	R	S
Weak	Yes					
Büchi						
Co-Büchi						
Parity						
Rabin			No		Y	N
Streett		N	Y	No		Y

Table 2 The state blow-up involved in the complementation of GFG automata.

Comp. To / From	$\overline{\text{W}}$	$\overline{\text{C}}$	$\overline{\text{B}}$	$\overline{\text{P}}$	$\overline{\text{R}}$	$\overline{\text{S}}$
Weak	Poly					
Büchi						
Co-Büchi			Exp			?
Parity						
Rabin						
Streett						

- From Streett to weak. Consider a GFG-NSW $\mathcal{A}$ that can be complemented to a GFG-NWW. Then the language of $\mathcal{A}$ is DWW-recognizable. Thus, by Theorems 10 and 15, there is a GFG-NWW on a substructure of $\mathcal{A}$, and we are back in case (1).
- From Streett to co-Büchi. Given a DRW $\mathcal{A}$ that is NCW realizable, one can translate it to an equivalent NCW by first dualizing $\mathcal{A}$ into a DSW $\overline{\mathcal{A}}$ for the complement language, and then complementing $\overline{\mathcal{A}}$ into a GFG-NCW $\mathcal{C}$. Since dualizing a DRW into a DSW is done with no state blowup and the translation of DRWs to NCWs might involve an exponential state blowup [1], so does the complementation of GFG-NSW to GFG-NCWs.
- From Streett to Streett. Analogous to the above case of Streett to co-Büchi, due to the exponential state blowup in the translation of DRWs to NSWs [1]. ◀

Translating LTL formulas to GFG Automata

Recall that GFG-NCWs are exponentially more succinct than DCWs [6], suggesting they do have some power of nondeterministic automata. A natural question is whether one can come up with an exponential translation of LTL formulas to GFG automata, in particular when attention is restricted to LTL formulas that are DCW-realizable. We complete this section with a negative answer, providing another evidence for the deterministic nature of GFG automata. This result is based on the fact that the language with which the doubly-exponential lower bound of the translation of LTL to DBW in [9] is proven is bounded (that is, it is both safe and co-safe). It means that by Corollary 18, any GFG-NSW for it would be DetByP, contradicting the doubly-exponential lower bound.

▶ **Corollary 21.** *The translation of DCW-realizable LTL formulas into GFG-NSW is doubly exponential.*

References

1 U. Boker. Rabin vs. Streett automata. In *Proc. 37th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 17:1–17:15, 2017.

2 U. Boker, D. Kuperberg, O. Kupferman, and M. Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 89–100, 2013.

3 T. Colcombet and C. Löding. Regular cost functions over finite trees. In *Proc. 25th IEEE Symp. on Logic in Computer Science*, pages 70–79, 2010.

4 T.A. Henzinger and N. Piterman. Solving games without determinization. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006.

5 S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1994.

6 D. Kuperberg and M. Skrzypczak. On determinisation of Good-For-Games automata. In *Proc. 42nd Int. Colloq. on Automata, Languages, and Programming*, pages 299–310, 2015.

7 O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. *International Journal on the Foundations of Computer Science*, 17(4):869–884, 2006.

8 O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006.

9 O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

10 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

11 L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

12 G. Morgenstern. Expressiveness results at the bottom of the ω-regular hierarchy. M.Sc. Thesis, The Hebrew University, 2003.

13 D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd IEEE Symp. on Logic in Computer Science*, pages 422–427, 1988.

14 D. Niwiński and I. Walukiewicz. Relating hierarchies of word and tree automata. In *Proc. 15th Symp. on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*. Springer, 1998.

15 M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

16 S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

17 H. Seidl and D. Niwiński. On distributive fixed-point expressions. *Theoretical Informatics and Applications*, 33(4–5):427–446, 1999.

18 W. Thomas. On the synthesis of strategies in infinite games. In E.W. Mayr and C. Puech, editors, *Proc. 12th Symp. on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.

19 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

Popular Matchings with Multiple Partners*

Florian Brandl[1] and Telikepalli Kavitha[2]

1 **Technische Universität München, Germany**
`brandlfl@in.tum.de`
2 **Tata Institute of Fundamental Research, India**
`kavitha@tcs.tifr.res.in`

Abstract

Our input is a bipartite graph $G = (A \cup B, E)$ where each vertex in $A \cup B$ has a preference list strictly ranking its neighbors. The vertices in A and in B are called *students* and *courses*, respectively. Each student a seeks to be matched to $\mathsf{cap}(a) \geq 1$ many courses while each course b seeks $\mathsf{cap}(b) \geq 1$ many students to be matched to it. The Gale-Shapley algorithm computes a pairwise-stable matching (one with no blocking edge) in G in linear time. We consider the problem of computing a *popular* matching in G – a matching M is popular if M cannot lose an election to any matching where vertices cast votes for one matching versus another. Our main contribution is to show that a max-size popular matching in G can be computed by the *2-level Gale-Shapley* algorithm in linear time. This is an extension of the classical Gale-Shapley algorithm and we prove its correctness via linear programming.

1998 ACM Subject Classification G.2.2 Graph algorithms, G.1.6 Linear programming

Keywords and phrases Bipartite graphs, Linear programming duality, Gale-Shapley algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.19

1 Introduction

We study the many-to-many matching problem in bipartite graphs: formally, this is given by a set A of vertices (these vertices are called *students*) and a set B of vertices (these are called *courses*), where every vertex u has a capacity $\mathsf{cap}(u) \geq 1$. Every student a seeks to be matched to $\mathsf{cap}(a)$ many courses and every course b seeks $\mathsf{cap}(b)$ many students to be matched to it. Moreover, every student $a \in A$ has a strict ranking $\succ_a$ over courses that are acceptable to a and every course b has a strict ranking $\succ_b$ over students that are acceptable to b. The set of mutually acceptable pairs is given by $E \subseteq A \times B$. Thus our input is a bipartite graph $G = (A \cup B, E)$ and the preferences of a vertex are expressed as an ordered list of its neighbors, e.g., $u\colon v, v'$ denotes the preference $v \succ_u v'$, i.e., u prefers v to v'.

▶ **Definition 1.** A matching M in $G = (A \cup B, E)$ is a subset of E such that $|M(u)| \leq \mathsf{cap}(u)$ for each $u \in A \cup B$, where $M(u) = \{v : (u, v) \in M\}$.

The goal is to compute an *optimal* matching in G. The usual definition of optimality in this setting has been *pairwise-stability* [26]. A matching M in G is said to be pairwise-stable if there is no student-course pair (a, b) that "blocks" M. We say a pair $(a, b) \in E \setminus M$ blocks M if (1) either a has less than $\mathsf{cap}(a)$ partners in M or a prefers b to its least preferred neighbor in $M(a)$ and (2) either b has less than $\mathsf{cap}(b)$ partners in M or b prefers a to its least preferred

* The full version of this paper is available at `http://arxiv.org/abs/1609.07531`.

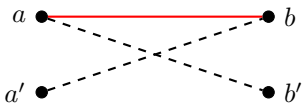
37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 19; pp. 19:1–19:15

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

neighbor in $M(b)$. It is known that pairwise-stable matchings always exist [26] and the Gale-Shapley algorithm [8] can be easily generalized to find such a matching in $G = (A \cup B, E)$. The many-to-one variant of this problem, also called the *hospitals/residents* problem, where $\mathsf{cap}(a) = 1$ for every $a \in A$, was studied by Gale and Shapley [8] who showed that their algorithm for the *marriage* problem (where $\mathsf{cap}(u) = 1$ for all $u \in A \cup B$) can be easily extended to find a stable matching in the hospitals/residents problem as well.

Since a (pairwise) stable matching is a maximal matching in G, its size is at least $|M_{\max}|/2$, where $M_{\max}$ is a max-size matching in G. This bound can be tight as shown by the following simple example: let $A = \{a, a'\}$ and $B = \{b, b'\}$ where each vertex has capacity 1 and the edge set is $E = \{(a, b), (a, b'), (a', b)\}$. The preferences are shown in the table below. Here the only stable matching (red line) is $S = \{(a, b)\}$, which is of size 1. However, the max-size matching (dashed lines) $M_{\max} = \{(a', b), (a, b')\}$ is of size 2.

a: b, b'	b: a, a'
a': b	b': a



It can be shown that all pairwise-stable matchings have to match the same set of vertices and every vertex gets matched to the same capacity in every pairwise-stable matching. In the hospitals/residents setting, this is popularly called the "Rural Hospitals Theorem" [9, 27]. More precisely, Roth [27] showed that not only is every hospital matched to the same number of residents in every stable matching, but moreover, every hospital that is not matched up to its capacity in some stable matching is actually matched to the same *set of residents* in any stable matching. Thus the notion of stability is very restrictive.

From a social point of view, it seems desirable to have a high number of students registered for courses to make effective use of available resources. Similarly, in the hospitals/residents setting, it seems desirable to have a higher number of residents matched to hospitals in order to keep few residents unemployed and guarantee sufficient staffing for hospitals. The latter point particularly applies to rural hospitals that oftentimes face the problem of being understaffed with residents by the National Resident Matching Program in the USA (cf. [25, 27]). This motivates relaxing the notion of "absence of blocking edges" to a weaker notion of stability so as to obtain matchings that are guaranteed to be significantly larger than $|M_{\max}|/2$. Note that we do not wish to ignore the preferences of vertices and impose a max-size matching on them as such a way of assignment may be highly undesirable from a social viewpoint. Instead our approach is to replace the *local* stability notion of "no blocking edges" with a weaker notion of *global* stability that achieves more "global good" in the sense that its size is always at least $\gamma \cdot |M_{\max}|$ for some $\gamma > 1/2$.

Popularity

To this end, we consider the notion of popularity, which was introduced by Gärdenfors [10] for the stable marriage problem: the input here consists of a set of men and a set of women, where each person seeks to get matched to at most one person from the opposite sex. Hence the marriage or the one-to-one matching setting, where every vertex has capacity 1, is a special case of the many-to-many matching setting considered here. Popularity is based on voting by vertices on the set of feasible matchings. In the one-to-one setting, the preferences of a vertex over its neighbors are extended to preferences over matchings by postulating that every vertex orders matchings in the order of its partners in these matchings. This postulates that vertices do not care which other pairs are formed.

A matching is popular if it never loses a head-to-head election against any matching where each vertex casts a vote. Thus popular matchings are (weak) *Condorcet winners* [5] in the corresponding voting instance. The Condorcet paradox shows that collective preferences can be cyclic and so there need not be a Condorcet winner; this is the source of many impossibility results in social choice theory such as Arrow's impossibility theorem.

However, in the context of matchings in the one-to-one setting, Gärdenfors [10] showed that every stable matching is popular. Hence the fact that stable matchings always exist here [8] implies that popular matchings always exist. This is quite remarkable given that popular matchings correspond to (weak) Condorcet winners. In the one-to-one setting, there is a vast literature on popular matchings [3, 15, 19, 13, 7, 20, 16].

Here we generalize the notion of popularity to the many-to-many matching setting. This requires us to specify how vertices vote over different subsets of their neighbors. In particular, one may want to allow a single vertex to cast multiple votes if its capacity is greater than 1. Our definition of voting by a vertex between two subsets of its neighbors is the following: first remove all vertices that are contained in both sets; then find a bijection from the first set to the second set and compare every vertex with its image under this bijection (if the sets are not of equal size, we add dummy vertices that are less preferred to all non-dummy vertices); the number of wins minus the number of losses is cast as the vote of the vertex. The vote may depend on the bijection that is chosen, however.

Our definition is based on the bijection that *minimizes* the vote, which results in a rather restrictive notion of popularity. We show however that even for this notion of popularity, every stable matching is popular. In particular, popular matchings always exist. As a consequence, popular matchings always exist for every notion of popularity that is less restrictive than our notion of popularity. Our goal is to find a *max-size* popular matching and crucially, it turns out that the size of a max-size popular matching is independent of the bijection that is chosen for the definition of popularity. We formalize these notions below.

In the one-to-one setting, given any two matchings M_0, M_1 and a vertex u, we say u prefers M_0 to M_1 if u prefers $M_0(u)$ to $M_1(u)$, where $M_i(u)$ is u's partner in M_i, for $i = 0, 1$, and we say "$M_i(u) = \text{null}$" if u is left unmatched in matching M_i – note that the null option is the least preferred state for any vertex. Define the function $\mathsf{vote}_u(v, v')$ for any vertex u and neighbors v, v' of u as follows: $\mathsf{vote}_u(v, v')$ is 1 if u prefers v to v', it is -1 if u prefers v' to v, and it is 0 otherwise (i.e., if $v = v'$). In the one-to-one setting, $\Delta_u(M_0, M_1)$, which is u's vote for M_0 versus M_1, is defined to be $\mathsf{vote}_u(M_0(u), M_1(u))$.

In the many-to-many setting, while comparing one matching with another, we allow a vertex to cast more than one vote. For instance, when we compare the preference of vertex u with $\mathsf{cap}(u) = 3$ for $S_0 = \{v_1, v_2, v_3\}$ versus $S_1 = \{v_4, v_5, v_6\}$ (where $v_1 \succ_u v_2 \succ_u \cdots \succ_u v_6$), we would like u's vote to capture the fact that u is better-off by *3 partners* in S_0 when compared to S_1. So we define u's vote for S_0 versus S_1 as follows. Let S_0, S_1 be any two subsets of the set of u's neighbors where we add some occurrences of "null" to the smaller of S_0, S_1 to make the two sets of the same size. We will view the sets $S_0' = S_0 \setminus S_1$ and $S_1' = S_1 \setminus S_0$ as arrays $\langle S_i'[1], \ldots, S_i'[k]\rangle$ (for $i = 0, 1$) where $k = |S_0| - |S_0 \cap S_1| = |S_1| - |S_0 \cap S_1|$. The preference of vertex u for S_0 versus S_1, denoted by $\delta_u(S_0, S_1)$, is defined as follows:

$$\delta_u(S_0, S_1) = \min_{\sigma \in \Pi[k]} \sum_{i=1}^{k} \mathsf{vote}_u(S_0'[i], S_1'[\sigma(i)]), \tag{1}$$

where $\Pi[k]$ is the set of permutations on $\{1, \ldots, k\}$. Let $\Delta_u(M_0, M_1) = \delta_u(S_0, S_1)$, where $S_0 = M_0(u)$ and $S_1 = M_1(u)$. So $\Delta_u(M_0, M_1)$ counts the number of votes by u for $M_0(u)$ versus $M_1(u)$ when the sets $S_0' = M_0(u) \setminus M_1(u)$ and $S_1' = M_1(u) \setminus M_0(u)$ are being compared

in the order that is most adversarial or *negative* for M_0. That is, this order $\sigma \in \Pi[k]$ of comparison between elements of S'_0 and S'_1 gives the least value for $n^+ - n^-$, where n^+ is the number of indices i such that $S'_0[i] \succ_u S'_1[\sigma(i)]$ and n^- is the number of indices i such that $S'_0[i] \prec_u S'_1[\sigma(i)]$. Note that $\Delta_u(M_0, M_1) + \Delta_u(M_1, M_0) \leq 0$ and it can be *strictly negative*.

For instance, when a vertex u with $\mathsf{cap}(u) = 3$ compares two subsets $S_0 = \{v_1, v_3, v_5\}$ and $S_1 = \{v_2, v_4, v_6\}$ (where $v_1 \succ_u v_2 \succ_u \cdots \succ_u v_6$), we have $\delta_u(S_0, S_1) = -1$ since comparing the following pairs results in the least value of $\delta_u(S_0, S_1)$: this pairing is (v_1 with v_6), (v_3 with v_2), (v_5 with v_4). This makes $\delta_u(S_0, S_1) = 1 - 1 - 1 = -1$. While computing $\delta_u(S_1, S_0)$, the pairing would be (v_2 with v_1), (v_4 with v_3), (v_6 with v_5): then $\delta_u(S_1, S_0) = -1 - 1 - 1 = -3$.

For any two matchings M_0 and M_1 in G, we compare them using the function $\Delta(M_0, M_1)$ defined as follows:

$$\Delta(M_0, M_1) = \sum_{u \in A \cup B} \Delta_u(M_0, M_1). \tag{2}$$

We say M_0 is at least as popular as M_1 if $\Delta(M_0, M_1) \geq 0$ and M_0 is more popular than M_1 if $\Delta(M_0, M_1) > 0$. If $\Delta(M_0, M_1) \geq 0$ then for every vertex u in $A \cup B$: *no matter in which order* the elements of $S'_0 = M_0(u) \setminus M_1(u)$ and $S'_1 = M_1(u) \setminus M_0(u)$ are compared against each other by u in the evaluation of $\Delta_u(M_0, M_1)$, when we sum up the total number of votes cast by all vertices, the votes for M_1 can *never* outnumber the votes for M_0.

▶ **Definition 2.** M_0 is a popular matching in $G = (A \cup B, E)$ if $\Delta(M_0, M_1) \geq 0$ for all matchings M_1 in G.

Thus for a matching M_0 to be popular, it means that M_0 is at least as popular as every matching in G, i.e., there is no matching M_1 with $\Delta(M_0, M_1) < 0$. If there exists a matching M_1 such that $\Delta(M_0, M_1) < 0$ then this is taken as a certificate of *unpopularity* of M_0. Note that it is possible that both $\Delta(M_0, M_1)$ and $\Delta(M_1, M_0)$ are negative, i.e., for each vertex u there is some order of comparison between the elements of $S'_0 = M_0(u) \setminus M_1(u)$ with those in $S'_1 = M_1(u) \setminus M_0(u)$ so that when we sum up the total number of votes cast by all the vertices, the number for M_1 is more than the number for M_0; similarly for each u there is another order of comparison between the elements of S'_0 with those in S'_1 so that when we sum up the total number of votes cast by all the vertices, the number for M_0 is more than the number for M_1. In this case neither M_0 nor M_1 is popular. It is not obvious whether popular matchings always exist in G.

Our definition of popularity may seem too strict and restrictive since for each vertex u, we choose the most negative or adversarial ordering for $M_0(u) \setminus M_1(u)$ versus $M_1(u) \setminus M_0(u)$ while calculating $\Delta_u(M_0, M_1)$. A more relaxed definition may be to order the sets $S'_0 = M_0(u) \setminus M_1(u)$ and $S'_1 = M_1(u) \setminus M_0(u)$ in increasing order of preference of u and take $\sum_i \mathsf{vote}_u(S'_0[i], S'_1[i])$ as u's vote. An even more relaxed definition may be to choose the most favorable or *positive* ordering for S'_0 versus S'_1 while calculating $\Delta_u(M_0, M_1)$. Note that as per (1) we have:

$$-\Delta_u(M_0, M_1) = -\min_{\sigma \in \Pi[k]} \sum_{i=1}^{k} \mathsf{vote}_u(S'_0[i], S'_1[\sigma(i)]) = \max_{\pi \in \Pi[k]} \sum_{i=1}^{k} \mathsf{vote}_u(S'_1[i], S'_0[\pi(i)]). \tag{3}$$

▶ **Definition 3.** Call a matching M_1 *weakly popular* if $\Delta(M_0, M_1) \leq 0$, i.e., $-\Delta(M_0, M_1) \geq 0$, for all matchings M_0 in G.

Thus it follows from (3) that M_1 is a weakly popular matching if the sum of votes for M_1 is at least the sum of votes for any matching M_0 when each vertex u compares $M_1(u) \setminus M_0(u)$ versus $M_0(u) \setminus M_1(u)$ in the ordering (as given by π) that is most favorable for M_1. In the

one-to-one setting, we have $\Delta(M_0, M_1) + \Delta(M_1, M_0) = 0$ for any pair of matchings M_0, M_1 since $\Delta_u(M_0, M_1) = \mathsf{vote}_u(M_0(u), M_1(u)) = -\mathsf{vote}_u(M_1(u), M_0(u)) = -\Delta_u(M_1, M_0)$ for each u; thus the notions of "popularity" and "weak popularity" coincide here. In the many-to-many setting, weak popularity is a more relaxed notion than popularity.

We choose a strong definition of popularity so that a matching that is popular according to our notion will also be popular according to any notion "in between" between popularity and weak popularity. However this breadth may come at a price as it could be the case that a max-size weakly popular matching is larger than a max-size popular matching.

Our results and techniques

We will show that every pairwise-stable matching in $G = (A \cup B, E)$ is popular, thus our (seemingly strong) definition of popularity is a relaxation of pairwise-stability. We will present a simple linear time algorithm for computing a max-size popular matching M_0 in G and show that $|M_0| \geq \frac{2}{3} \cdot |M_{\max}|$.

We also show that M_0 is more popular than every *larger* matching, i.e., $\Delta(M_0, M_1) > 0$ (refer to (2)) for any matching M_1 that is larger than M_0. Thus M_0 is also a *max-size weakly popular matching* in G as no matching M_1 larger than M_0 can be weakly popular due to the fact that $\Delta(M_0, M_1) > 0$. Thus surprisingly, we lose nothing in terms of the size of our matching by sticking to a strong notion of popularity.

Akin to the rural hospitals theorem, we show that all max-size popular matchings have to match the same set of vertices and every vertex gets matched to the same capacity in every max-size popular matching. However, even in the hospitals/residents setting, hospitals that are not matched up to their capacity in some max-size popular matching do *not* need to be matched to the same sets of residents in any max-size popular matching, which is in contrast to stable matchings [27].

Our algorithm is an extension of the 2-level Gale-Shapley algorithm from [19] to find a max-size popular matching in a stable marriage instance. While the analysis of the 2-level Gale-Shapley algorithm in [19] is based on a structural characterization of popular matchings (from [15]) on forbidden alternating paths and alternating cycles, we use linear programming here to show a simple and self-contained proof of correctness of our algorithm. We would like to remark that the structural characterization from [15] and the proof of correctness from [19] can be extended (in a rather laborious manner) to show the correctness of the generalized algorithm in our more general setting as well, however our proof of correctness is much simpler and this yields a much easier proof of correctness of the algorithm in [19]. Our linear programming techniques are based on a linear program used in [21] to find a popular fractional matching in a bipartite graph with *1-sided preference lists*.

Background and related work

The first algorithmic question studied for popular matchings was in the domain of 1-sided preference lists [1] where it is only vertices on the left, who are *agents*, that have preferences; the vertices on the right are *objects* and they have no preferences. Popular matchings need not always exist here, however fractional matchings that are popular always exist and can be computed in polynomial time via linear programming [21]. Popular matchings always exist in any instance of the stable marriage problem with strict preference lists since every stable matching is popular [10].

Efficient algorithms to find a max-size popular matching in a stable marriage instance are known [15, 19] and a subclass of max-size popular matchings called dominant matchings

was studied in [7] and it was shown that these matchings coincide with stable matchings in a larger graph. A polynomial time algorithm was shown in [20] to find a min-cost popular half-integral matching when there is a cost function on the edge set and it was shown in [16] that the popular fractional matching polytope here is half-integral. When preference lists admit ties, the problem of determining if a stable marriage instance $(A \cup B, E)$ admits a popular matching or not is NP-hard [3] and the NP-hardness of this problem holds even when ties are allowed on only one side (say, in the preference lists of vertices in A) [6].

Very recently and independent of our work, the problem of computing a max-size popular matching in an extension of the hospitals/residents problem, i.e., in the *many-to-one* setting, was considered by Nasre and Rawat [23]. The notion of "more popular than" in [23] is weaker than ours: in order to compare matchings M_0 and M_1, in [23] every hospital h orders $S'_0 = M_0(h) \setminus M_1(h)$ and $S'_1 = M_1(h) \setminus M_0(h)$ in increasing order of preference of h and $\sum_i \mathsf{vote}_h(S'_0[i], S'_1[i])$ is h's vote for M_0 versus M_1. An efficient algorithm was shown for their problem by reducing it to a stable matching problem in a larger graph – this closely follows the method and techniques in [15, 19, 7] for the max-size popular matching problem in the one-to-one setting. Note that popularity as per their definition is "in between" our notions of popularity and weak popularity.

The stable matching problem in a marriage instance has been extensively studied – we refer to the books [11, 22] on this topic. The problem of computing stable matchings or its variants in the hospitals/residents setting is also well-studied [14, 2, 12, 17, 18]. The stable matching algorithm in the hospitals/residents problem has several real-world applications: it is used to match residents to hospitals in Canada [28] and in the USA [24]. The many-to-many stable matching problem has also received considerable attention [26, 4, 29].

2 Our algorithm

A first attempt to solve the max-size popular matching problem in a many-to-many instance $G = (A \cup B, E)$ may be to define an equivalent one-to-one instance $G' = (A' \cup B', E')$ by making $\mathsf{cap}(u)$ copies of each $u \in A \cup B$ and $\mathsf{cap}(a) \cdot \mathsf{cap}(b)$ many copies of each edge (a, b); the max-size popular matching problem in G' can be solved using the algorithm in [19] and the obtained matching $\tilde{M}$ in G' can be mapped to a matching M in G. In the first place, one should ensure that there are no multi-edges in M. The matching M will be popular, however it is not obvious that M is a *max-size* popular matching in G as every popular matching in G need not be realized as some popular matching in G': we show such an example in the Appendix. Thus one needs to show that there is at least one max-size popular matching in G that can be realized as a popular matching in G'; we do not pursue this approach here as the running time of the max-size popular matching algorithm in G' is linear in the size of G', which is $O(mn)$, where $|E| = m$ and $|A| + |B| = n$.

In this section we describe a simple $O(m + n)$ algorithm called the generalized *2-level Gale-Shapley algorithm* to compute a max-size popular matching in $G = (A \cup B, E)$. This algorithm works on the graph $H = (A'' \cup B, E'')$ defined as follows: A'' consists of two copies a^0 and a^1 of every student a in A, i.e., $A'' = \{a^0, a^1 : a \in A\}$. The set B of courses in H is the same as in G and the edge set here is $E'' = \{(a^0, b), (a^1, b) : (a, b) \in E\}$.

The preference list of a^i (for $i = 0, 1$) is exactly the same as the preference list of a. The elements in the set $\{a^i : a \in A\}$ will be called *level* i students, for $i = 0, 1$. Every $b \in B$ prefers any level 1 neighbor to any level 0 neighbor: within the set of level i neighbors (for $i = 0, 1$), b's preference order is the same as its original preference order. For instance, if a course b has only 2 neighbors a and v in G where $a \succ_b v$, the preference order of b in

Algorithm 1 *Input:* $H = (A'' \cup B, E'')$; *Output:* *A matching* M *in* H

1. Initialize $Q = \{a^0 : a \in A\}$ and $M = \emptyset$. Set $\mathsf{residual}(a) = \mathsf{cap}(a)$ for all $a \in A$.
2. **while** $Q \neq \emptyset$ **do**
3. delete the first vertex from Q: call this vertex a^i.
4. **while** a^i has one or more neighbors in H to propose to and $\mathsf{residual}(a) > 0$ **do**
5. – let b be the most preferred neighbor of a^i in H that a^i has not yet proposed to. {*So every neighbor of* a^i *in the current graph* H *that is ranked better than* b *is already matched to* a^i *in* M.}
6. – add the edge (a^i, b) to M.
7. **if** $i = 1$ and b is already matched to a^0 **then**
8. – delete the edge (a^0, b) from M. {*So* (a^0, b) *in* M *gets replaced by* (a^1, b).}
9. **else**
10. – set $\mathsf{residual}(a) = \mathsf{residual}(a) - 1$. {*since* $|M(a)|$ *has increased by 1*}
11. **if** b is matched to more than $\mathsf{cap}(b)$ partners in M **then**
12. – let v^j be b's worst partner in M. Delete the edge (v^j, b) from M. {*Note that "worst" is as per preferences in* H.}
13. – set $\mathsf{residual}(v) = \mathsf{residual}(v) + 1$ and if $v^j \notin Q$ then add v^j to Q.
14. **end if**
15. **end if**
16. **if** b is matched to $\mathsf{cap}(b)$ many partners in M **then**
17. – delete all edges (u, b) from H where u is a neighbor of b in H that is ranked worse than b's worst partner in M. {*"Worse" is as per preferences in* H.}
18. **end if**
19. **end while**
20. **if** $\mathsf{residual}(a) > 0$ and $i = 0$ **then**
21. – add a^1 to Q. {*Though* $\mathsf{residual}(a) > 0$, *the condition in the above while-loop does not hold, i.e.,* a^0 *has no neighbors in* H *to propose to; hence* a^1 *gets activated.*}
22. **end if**
23. **end while**
24. Return the matching M.

G' is: a^1, v^1, a^0, v^0. The *sum* of capacities of a^0 and a^1 will be $\mathsf{cap}(a)$ and we will use $\mathsf{residual}(a)$ to denote the $\mathsf{cap}(a) - |M(a)|$, where M is the current matching. At any point in time, only one of a^0 and a^1 will be *active* in our algorithm.

A description of our algorithm is given as Algorithm 1. To begin with, all level 0 students are active in our algorithm and all level 1 students are inactive. We keep a queue Q of all the active students and they propose as follows:

- every active student a^i, where a is not fully matched, proposes to its most preferred neighbor in H that it has not yet proposed to (lines 4-5 of Algorithm 1)
- if a^0 has already proposed to all its neighbors in H and a is not fully matched, then a^0 becomes inactive and a^1 becomes active and it joins the queue Q (lines 20-21).

When a course b receives a proposal from a^i, the vertex b accepts this offer (in line 6). In case b is already matched to a^0 and it now received a proposal from a^1, the edge (a^0, b) in M is replaced by the edge (a^1, b) (otherwise b would end up being matched to a with multiplicity 2 which is not allowed) – this is done in lines 7-8 of Algorithm 1.

If b is now matched to more than $\mathsf{cap}(b)$ partners then b rejects its worst partner v^j in the current matching and so $\mathsf{residual}(v)$ increases by 1 and v^j joins Q if it is not already in Q

(in lines 11-13). If b is now matched to $\mathsf{cap}(b)$ partners then we delete all edges (u, b) from H where u is a neighbor of b in H that is ranked worse than b's worst partner in the current matching – so no such resident u can propose to b later on in the algorithm (lines 16-17). Once Q becomes empty, the algorithm terminates.

Let M be the matching returned by this algorithm and let M_0 be the matching in G that is obtained by projecting M to the edge set of G, i.e., $(a, b) \in M_0$ if and only if $(a^i, b) \in M$ for some $i \in \{0, 1\}$. We will prove that M_0 is a max-size popular matching in Section 3.

3 The correctness of our algorithm

In this section we show a sufficient condition for a matching N in G to be popular. This is shown via a graph called G'_N: this is a bipartite graph constructed using N such that N gets mapped to a *simple matching* N' in G'_N, i.e., $|N'(v)| \leq 1$ for all vertices v in G'_N.

The vertex set of G'_N includes $A' \cup B'$ where $A' = \{a_i : a \in A \text{ and } 1 \leq i \leq \mathsf{cap}(a)\}$ and $B' = \{b_j : b \in B \text{ and } 1 \leq j \leq \mathsf{cap}(b)\}$. That is, for each vertex $u \in A \cup B$, there are $\mathsf{cap}(u)$ many copies of u in G'_N. For each edge (a, b) in G such that $(a, b) \in N$, we will arbitrarily choose a distinct $i \in \{1, \ldots, \mathsf{cap}(a)\}$ and a distinct $j \in \{1, \ldots, \mathsf{cap}(b)\}$ and include (a_i, b_j) in N'. If $u \in A \cup B$ was not fully matched in N, i.e., it has less than $\mathsf{cap}(u)$ many partners in N, then some u_k's will be left unmatched in N'.

1. For each edge (a, b) in G such that $(a, b) \notin N$, we will have edges (a_i, b_j) in G'_N, for all $1 \leq i \leq \mathsf{cap}(a)$ and $1 \leq j \leq \mathsf{cap}(b)$.
2. For each edge $(a, b) \in N$, we will have the edge (a_i, b_j) in G'_N where $(a_i, b_j) \in N'$.

Thus for any edge $e = (a, b) \notin N$, there are $\mathsf{cap}(a) \cdot \mathsf{cap}(b)$ many copies of e in G': these are (a_i, b_j) for all $(i, j) \in \{1, \ldots, \mathsf{cap}(a)\} \times \{1, \ldots, \mathsf{cap}(b)\}$. However for any edge $(a, b) \in N$, there is only *one* edge (a_i, b_j) in G'_N where $(a_i, b_j) \in N'$, in other words, the student a_i is not adjacent in G'_N to course $b_{j'}$ for $j' \neq j$ and similarly, the course b_j is not adjacent in G'_N to student $a_{i'}$ for $i' \neq i$. The Appendix has an example of G'_N corresponding to a matching N in a many-to-one instance G (see Fig. 2).

There are also some new vertices called "last resort neighbors" in G'_N: for any $u_k \in A' \cup B'$, we introduce a new vertex $\ell(u_k)$; the vertex u_k ranks $\ell(u_k)$ at the bottom of its preference list.

3. The edge set of G'_N also contains the edges $(u_k, \ell(u_k))$ for each $u_k \in A' \cup B'$.

The purpose of the vertex $\ell(u_k)$ is to capture the state of $u_k \in A' \cup B'$ being left unmatched in any matching so that every matching in G gets mapped to an $(A' \cup B')$-*complete matching* in G'_N, i.e., one that matches all vertices in $A' \cup B'$. We will use these last resort neighbors to obtain an $(A' \cup B')$-complete matching N^* from N'.

$$N^* = N' \cup \{(u_k, \ell(u_k)) : u_k \in A' \cup B' \text{ and } u_k \text{ is unmatched in } N'\}.$$

Thus if a vertex $u \in A \cup B$ was not fully matched in N, then some u_i's will be matched to their last resort neighbors in N^*. We now define edge weights in G'_N.

- For any edge $e = (a_i, b_j) \in A' \times B'$: the weight of edge e is $\mathsf{wt}_N(e) = \mathsf{vote}_a(b, N^*(a_i)) + \mathsf{vote}_b(a, N^*(b_j))$, where $N^*(u_k)$ is u_k's partner in the $(A' \cup B')$-complete matching N^*. Thus $\mathsf{wt}_N(a_i, b_j)$ is the sum of votes of a and b for each other versus $N^*(a_i)$ and $N^*(b_j)$, respectively. We have $\mathsf{wt}_N(e) \in \{\pm 2, 0\}$ and $\mathsf{wt}_N(e) = 2$ if and only if e blocks N.

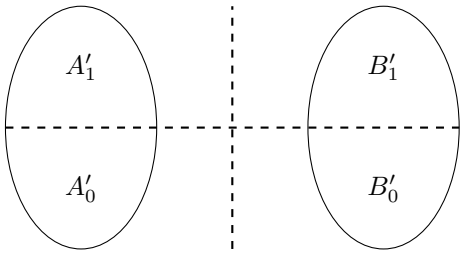


Figure 1 $A' = A'_0 \cup A'_1$ and $B' = B'_0 \cup B'_1$: all courses b_j left unmatched in M'_0 are in B'_0 and all students a_i left unmatched in M'_0 are in A'_1. Note that $M'_0 \subseteq (A'_0 \times B'_0) \cup (A'_1 \times B'_1)$.

- For any edge $e = (u_k, \ell(u_k))$ where $u_k \in A' \cup B'$: the weight of edge e is $\mathsf{wt}_N(e) = \mathsf{vote}_u(\ell(u_k), N^*(u_k))$. Thus $\mathsf{wt}_N(u_k, \ell(u_k)) = -1$ if the vertex u_k was matched in N' and $\mathsf{wt}_N(u_k, \ell(u_k)) = 0$ otherwise (in which case $N^*(u_k) = \ell(u_k)$).

Observe that every edge $e \in N^*$ satisfies $\mathsf{wt}_N(e) = 0$. Thus the weight of the matching N^* in G'_N is 0. Theorem 4 below states that if *every* $(A' \cup B')$-complete matching in the graph G'_N has weight at most 0, then N is a popular matching in G.

The proof of Theorem 4 (given in the full version of the paper) shows that for any matching T in G, we can construct a realization T^* of T in G'_N such that T^* is an $(A' \cup B')$-complete matching and $\mathsf{wt}_N(T^*) = -\Delta(N, T)$. Thus if every $(A' \cup B')$-complete matching in G'_N has weight at most 0, then $\mathsf{wt}_N(T^*) \leq 0$, in other words, $\Delta(N, T) \geq 0$. Since $\Delta(N, T) \geq 0$ for all matchings T in G, the matching N is popular.

▶ **Theorem 4.** *Let N be a matching in G such that every $(A' \cup B')$-complete matching in G'_N has weight at most 0. Then N is popular.*

▶ **Corollary 5.** *Every pairwise-stable matching in G is popular.*

Proof. Let S be any pairwise-stable matching in G. Consider the graph G'_S: since S has no blocking edge in G, every edge e in G'_S satisfies $\mathsf{wt}_S(e) \leq 0$. Thus every matching in G'_S has weight at most 0 and so by Theorem 4, we can conclude that S is popular. ◀

3.1 The popularity of M_0

We will now use Theorem 4 to prove the popularity of the matching M_0 computed in Section 2. We will construct the matchings M'_0, M^*_0 and the graph G'_{M_0} corresponding to the matching M_0 as described at the beginning of Section 3. Our goal is to show that every $(A' \cup B')$-complete matching in G'_{M_0} has weight at most 0. Note that the matching M^*_0 has weight 0 in G'_{M_0}.

We partition the set A' into $A'_0 \cup A'_1$ and the set B' into $B'_0 \cup B'_1$ as follows. Initialize $A'_0 = A'_1 = B'_0 = B'_1 = \emptyset$. For each edge $(a_i, b_j) \in M'_0$ do:

- if $(a^0, b) \in M$ then add a_i to A'_0 and b_j to B'_0;
- else (i.e., $(a^1, b) \in M$) add a_i to A'_1 and b_j to B'_1.

Recall that $M \subseteq A'' \times B$ is the matching in the graph H obtained at the end of the 2-level Gale-Shapley algorithm (see Algorithm 1) and the projection of M on to $A \times B$ is M_0.

The definition of the sets A'_0, A'_1, B'_0, B'_1 implies that $M'_0 \subseteq (A'_0 \times B'_0) \cup (A'_1 \times B'_1)$. Also add students unmatched in M'_0 to A'_1 and courses unmatched in M'_0 to B'_0. Thus we have $A' = A'_0 \cup A'_1$ and $B' = B'_0 \cup B'_1$ (see Fig. 1).

Theorem 6 will show that the matching M_0 satisfies the condition of Theorem 4, this will prove that M_0 is a popular matching in G. This proof is inspired by the proof in [20] that shows the membership of certain half-integral matchings in the popular fractional matching polytope of a stable marriage instance.

In order to show that every $(A' \cup B')$-complete matching in G'_{M_0} has weight at most 0, we consider the max-weight $(A' \cup B')$-complete matching problem in G'_{M_0} as our primal LP. We show a dual feasible solution $\vec{\alpha}$ that makes the dual objective function 0. This means the primal optimal value is at most 0 and this is what we set out to prove.

▶ **Theorem 6.** *Every $(A' \cup B')$-complete matching in G'_{M_0} has weight at most 0.*

Proof. Let our primal LP be the max-weight $(A' \cup B')$-complete matching problem in G'_{M_0}. We want to show that the primal optimal value is at most 0. The primal LP is the following:

$$\max \sum_{e \in G'_{M_0}} \mathsf{wt}_{M_0}(e) \cdot x_e$$

$$\text{subject to} \quad \sum_{e \in E'(u_k)} x_e = 1 \qquad \text{for all } u_k \in A' \cup B',$$

$$x_e \geq 0 \qquad \text{for all edges } e \in G'_{M_0},$$

where $E'(u_k)$ is the set of edges incident on u_k in G'_{M_0}.

The dual LP is the following: we associate a variable α_{u_k} to each vertex $u_k \in A' \cup B'$.

$$\min \sum_{u_k \in A' \cup B'} \alpha_{u_k}$$

$$\text{subject to} \quad \alpha_{a_i} + \alpha_{b_j} \geq \mathsf{wt}_{M_0}(a_i, b_j) \qquad \text{for all edges } (a_i, b_j) \in G'_{M_0}, \tag{4}$$

$$\alpha_{u_k} \geq \mathsf{wt}_{M_0}(u_k, \ell(u_k)) \qquad \text{for all } u_k \in A' \cup B'. \tag{5}$$

Consider the following assignment of α-values for all $u_k \in A' \cup B'$: set $\alpha_{u_k} = 0$ for all u_k unmatched in M'_0 (each such vertex is in $A'_1 \cup B'_0$) and for the matched vertices u_k in M'_0, we set α-values as follows: $\alpha_{u_k} = 1$ if $u_k \in A'_0 \cup B'_1$ and $\alpha_{u_k} = -1$ if $u_k \in A'_1 \cup B'_0$.

Observe that Inequality (5) holds for all vertices $u_k \in A' \cup B'$. This is because $\alpha_{u_k} = 0 = \mathsf{wt}_{M_0}(u_k, \ell(u_k))$ for all u_k unmatched in M'_0; similarly, for all u_k matched in M'_0 we have $\alpha_{u_k} \geq -1 = \mathsf{wt}_{M_0}(u_k, \ell(u_k))$. In order to show Inequality (4), we will use Claim 7 stated below – its proof follows from our algorithm and is included in the full version.

▶ **Claim 7.** *Let $e = (a_i, b_j)$ be any edge in G'_{M_0}.*

(i) *If $e \in A'_1 \times B'_0$, then $\mathsf{wt}_{M_0}(e) = -2$.*

(ii) *If $e \in (A'_0 \times B'_0) \cup (A'_1 \times B'_1)$, then $\mathsf{wt}_{M_0}(e) \leq 0$.*

- Claim 7 (i) says that for every edge $(a_i, b_j) \in A'_1 \times B'_0$ in G'_{M_0}, we have $\mathsf{wt}_{M_0}(a_i, b_j) = -2$. Since $\alpha_{u_k} \geq -1$ for all $u_k \in A'_1 \cup B'_0$, Inequality (4) holds for all edges of G'_{M_0} in $A'_1 \times B'_0$.
- Claim 7 (ii) says that for every edge (a_i, b_j) in $(A'_0 \times B'_0) \cup (A'_1 \times B'_1)$, we have $\mathsf{wt}_{M_0}(a_i, b_j) \leq 0$. Since $\alpha_{a_i} + \alpha_{b_j} \geq 0$ for all $(a_i, b_j) \in A'_t \times B'_t$ (for $t = 0, 1$), Inequality (4) holds for all edges of G'_{M_0} in $(A'_0 \times B'_0) \cup (A'_1 \times B'_1)$.

Since $\mathsf{wt}_{M_0}(e) \leq 2$ for all edges e in G'_{M_0} and we set $\alpha_{u_k} = 1$ for all vertices $u_k \in A'_0 \cup B'_1$, Inequality (4) is satisfied for all edges of G'_{M_0} in $A'_0 \times B'_1$. Thus Inequality (4) holds for all edges (a_i, b_j) in G'_{M_0} and so these α-values are dual feasible.

For every edge $(a_i, b_j) \in M'_0$, we have $\alpha_{a_i} + \alpha_{b_j} = 0$ and $\alpha_{u_k} = 0$ for vertices u_k unmatched in M'_0. Hence it follows that $\sum_{u_k \in A' \cup B'} \alpha_{u_k} = 0$. So by weak duality, the optimal value of the primal LP is at most 0. In other words, every matching in G'_{M_0} that matches all vertices in $A' \cup B'$ has weight at most 0. ◀

3.2 Maximality of the popular matching M_0

We need to show that M_0 is a max-size popular matching in G and we now show that this follows quite easily from the proof of Theorem 6. Let T be any matching in G. We can obtain a realization T^* of the matching T in G'_{M_0} that is absolutely analogous to how it was done to prove Theorem 4. Thus T^* is an $(A' \cup B')$-complete matching in G'_{M_0} and $\mathsf{wt}_{M_0}(T^*) = -\Delta(M_0, T)$.

We know from Theorem 6 that $\mathsf{wt}_{M_0}(T^*) \leq 0$. Suppose T is a popular matching in G. Then $\mathsf{wt}_{M_0}(T^*)$ has to be 0, otherwise the popularity of T is contradicted since $\mathsf{wt}_{M_0}(T^*) < 0$ implies that $\Delta(M_0, T) > 0$ (because $\mathsf{wt}_{M_0}(T^*) = -\Delta(M_0, T)$).

So if T is a popular matching in G, then T^* is an optimal solution to the maximum weight $(A' \cup B')$-complete matching problem in G'_{M_0}. Recall that this is the primal LP in the proof of Theorem 6. We will use the dual feasible solution $\vec{\alpha}$ that we constructed in the proof of Theorem 6 and apply complementary slackness to show that if $(u_k, \ell(u_k)) \in M_0^*$, i.e., if u_k is left unmatched in M'_0, then T^* also has to contain the edge $(u_k, \ell(u_k))$. This will imply that $|T| \leq |M_0|$, i.e., every popular matching in G has size at most $|M_0|$.

▶ **Lemma 8.** *Let T be a popular matching in G and let T^* be the realization of T in G'_{M_0}. Then for any vertex $u_k \in A' \cup B'$ we have: $(u_k, \ell(u_k)) \in M_0^*$ implies $(u_k, \ell(u_k)) \in T^*$.*

Proof. Consider the α-values assigned to vertices in $A' \cup B'$ in the proof of Theorem 6. This is an *optimal dual* solution since its value is 0 which is the value of the optimal primal solution. Thus complementary slackness conditions have to hold for each edge in the optimal solution $(T^*_e)_{e \in G'_{M_0}}$ to the primal LP. That is, for each edge $(u_k, v_t) \in G'_{M_0}$, we have:

$$\text{either} \quad \alpha_{u_k} + \alpha_{v_t} = \mathsf{wt}_{M_0}(u_k, v_t) \quad \text{or} \quad T^*_{(u_k, v_t)} = 0. \tag{6}$$

Let $u_k \in A' \cup B'$ be a vertex such that $(u_k, \ell(u_k)) \in M_0^*$, so $\alpha_{u_k} = 0$. If $u \in A$, then $u_k \in A'_1$. Observe that all of u_k's neighbors in G'_{M_0} are in B'_1 – this is because for any neighbor $v_t \neq \ell(u_k)$ of u_k, we have $\mathsf{vote}_u(v, \ell(u_k)) = 1$ and so $\mathsf{wt}_{M_0}(u_k, v_t) \geq 0$. Claim 7 (i) says that $\mathsf{wt}_{M_0}(u_k, v_t) = -2$ for all edges $(u_k, v_t) \in A'_1 \times B'_0$. Thus u_k has no neighbor in B'_0. Similarly, if $u \in B$, then $u_k \in B'_0$ and all its neighbors in G'_{M_0} are in A'_0; otherwise u_k has a neighbor v_t in A'_1 and Claim 7 (i) would get contradicted since $\mathsf{wt}_{M_0}(u_k, v_t) \geq 0$.

In both cases, every edge $(u_k, v_t) \in A' \times B'$ that is incident on u_k in G'_{M_0} is *slack* because $(u_k, v_t) \in (A'_0 \times B'_0) \cup (A'_1 \times B'_1)$: thus $\alpha_{u_k} = 0$ and $\alpha_{v_t} = 1$ while $\mathsf{wt}_{M_0}(u_k, v_t) = \mathsf{vote}_u(v, \ell(u_k)) + \mathsf{vote}_v(u, M'_0(v_t)) = 1 - 1 = 0$. Thus it follows from Equation (6) that $T^*_{(u_k, v_t)} = 0$ for $v_t \neq \ell(u_k)$. Since T^* is $(A' \cup B')$-complete, we have $(u_k, \ell(u_k)) \in T^*$. ◀

Now it immediately follows that M_0 is a max-size popular matching in G. Let T be any popular matching in G. Consider the matching $T' = T^* \setminus \{(u_k, \ell(u_k)) : u_k \in A' \cup B'\}$. Lemma 8 implies that $|T'| \leq |M'_0|$ because every vertex u_k left unmatched in M'_0 has to be left unmatched in T' also. Since $|T| = |T'|$ and $|M'_0| = |M_0|$, we have $|T| \leq |M_0|$. As this holds for any popular matching T in G, we can conclude that M_0 is a max-size popular matching in G.

Our algorithm can be easily implemented to run in linear time (the full version has these details). Hence we can conclude the following theorem.

▶ **Theorem 9.** *A max-size popular matching in a many-to-many instance $G = (A \cup B, E)$ can be computed in linear time.*

Lemma 10 (proved in the full version of the paper) states that no matching larger than M_0 can be weakly popular (see Definition 3) as $\Delta(M_0, T) > 0$ for any such matching T. This implies that M_0 is also a max-size weakly popular matching in G.

▶ **Lemma 10.** *Let T be a matching such that $|T| > |M_0|$. Then $\Delta(M_0, T) > 0$, i.e., M_0 is more popular than T.*

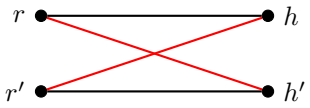
Interestingly, Lemma 10 implies that for any definition of popularity that is "in between" popularity and weak popularity, the size of a max-size popular matching is the same. To formalize the meaning of "in between", consider the two relations on matchings $\succsim_p$ and $\succsim_{wp}$, where $M_0 \succsim_p M_1$ if $\Delta(M_0, M_1) \geq 0$ and $M_0 \succsim_{wp} M_1$ if $\Delta(M_1, M_0) \leq 0$, induced by popularity and weak popularity, respectively. Clearly, $\succsim_p \subseteq \succsim_{wp}$. Note that popular matchings and weakly popular matchings correspond to maximal elements of $\succsim_p$ and $\succsim_{wp}$, respectively.[1] We showed that M_0, which is a max-size maximal element of $\succsim_p$, is also a max-size maximal element of $\succsim_{wp}$. This implies that if $\succsim$ is a relation on matchings (induced by an alternative notion of popularity) such that $\succsim_p \subseteq \succsim \subseteq \succsim_{wp}$, then M_0 is also a max-size maximal element of $\succsim$. This allows us to conclude the following proposition which even allows for different vertices to compare sets of neighbors in different ways.

▶ **Proposition 11.** The size of a max-size popular matching in $G = (A \cup B, E)$ is invariant to the way vertices compare sets of neighbors as long as it is in between the most adversarial and the most favorable comparison.

We now briefly discuss some other results that we show here. The rural hospitals theorem for stable matchings [27] does not necessarily hold for max-size popular matchings. That is, a hospital that is not matched up to capacity in some max-size popular matching is not necessarily matched to the same set of residents in every max-size popular matching.

Consider the instance $G = (R \cup H, E)$ with $R = \{r, r'\}$ and $H = \{h, h'\}$ and $\mathsf{cap}(h) = 1$ and $\mathsf{cap}(h') = 2$. The edge set is $R \times H$. The preferences are shown in the table below. The (max-size) popular matchings are $M = \{(r, h), (r', h')\}$ (in black) and $M' = \{(r, h'), (r', h)\}$ (in red). So h' is matched to a different resident in the two max-size popular matchings M and M'. Note that M' is not stable, as (r, h) is a blocking pair.

r: h, h'	h: r, r'
r': h, h'	h': r, r'



However Lemma 12 (proved in the full version) holds here. Such a result for max-size popular matchings in the one-to-one setting (that every max-size popular matching has to match the same set of vertices) was shown in [13]. Our proof is based on linear programming and is different from the combinatorial proof in [13].

▶ **Lemma 12.** *Let T be a max-size popular matching in G. Then T matches the same vertices as M_0 (the matching computed in Section 2) and moreover, every vertex u is matched in T to the same capacity as it gets matched to in M_0.*

The following results are also included in the full version. These proofs are inspired by analogous proofs in the one-to-one setting shown in [19] and in [15], respectively.

▶ **Lemma 13.** *We have $|M_0| \geq \frac{2}{3}|M_{\max}|$, where M_0 is a max-size popular matching in G and M_{max} is a max-size matching in G.*

▶ **Lemma 14.** *A pairwise-stable matching S is a min-size weakly popular matching in G.*

[1] M_0 is a maximal element of a relation $\succsim$ if for all elements M_1 we have: $M_1 \succsim M_0$ implies $M_0 \sim M_1$.

Acknowledgments. The first author wishes to thank Larry Samuelson for comments on the motivation for popular matchings. The second author wishes to thank David Manlove and Bruno Escoffier for asking her about popular matchings in the hospitals/residents setting.

References

1. D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.
2. G. Askalidis, N. Immorlica, A. Kwanashie, D. Manlove, and E. Pountourakis. Socially stable matchings in the hospitals/residents problem. In *the 13th International Symposium on Algorithms and Data Structures (WADS)*, pages 85–96, 2013.
3. P. Biró, R. W. Irving, and D. F. Manlove. Popular matchings in the marriage and roommates problems. In *the 7th International Conference on Algorithms and Complexity (CIAC)*, pages 97–108, 2010.
4. C. Blair. The lattice structure of the set of stable matchings with multiple partners. *Mathematics of Operations Research*, 13:619–628, 1988.
5. Marquis de Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie Royale, 1785. Facsimile published in 1972 by Chelsea Publishing Company, New York.
6. Á. Cseh, C.-C. Huang, and T. Kavitha. Popular matchings with two-sided preferences and one-sided ties. In *the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 367–379, 2015.
7. Á. Cseh and T. Kavitha. Popular edges and dominant matchings. In *the 18th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 138–151, 2016.
8. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
9. D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11(3):223–232, 1985.
10. P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Sciences*, 20(3):166–173, 1975.
11. D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Boston, MA, 1989.
12. K. Hamada, K. Iwama, and Shuichi Miyazaki. The hospitals/residents problem with lower quotas. *Algorithmica*, 74(1):440–465, 2016.
13. M. Hirakawa, Y. Yamauchi, S. Kijima, and M. Yamashita. On the structure of popular matchings in the stable marriage problem - who can join a popular matching? In the 3rd International Workshop on Matching Under Preferences (MATCH-UP), 2015.
14. C.-C. Huang. Classified stable matching. In *the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1235–1253, 2010.
15. C.-C. Huang and T. Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.
16. C.-C. Huang and T. Kavitha. Popularity, self-duality, and mixed matchings. In *the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2294–2310, 2017.
17. R. W. Irving, D. F. Manlove, and S. Scott. The hospitals/residents problem with ties. In *the 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 259–271, 2000.
18. R. W. Irving, D. F. Manlove, and S. Scott. Strong stability in the hospitals/residents problem. In *the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 439–450, 2003.
19. T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

20 T. Kavitha. Popular half-integral matchings. In *the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22.1–22.13, 2016.

21 T. Kavitha, J. Mestre, and M. Nasre. Popular mixed matchings. *Theoretical Computer Science*, 412(24):2679–2690, 2011.

22 D. F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific Publishing Company, 2013.

23 M. Nasre and A. Rawat. Popularity in the generalized hospital residents setting. In *the 12th International Computer Science Symposium in Russia (CSR)*, pages 245–259, 2017.

24 National Resident Matching Program. Why the match? Web document available at `http://www.nrmp.org/whythematch.pdf`.

25 A. E. Roth. The evolution of the labor market for medical interns and resident: a case study in game theory. *The Journal of Political Economy*, 92(6):991–1016, 1984.

26 A. E. Roth. Stability and polarization of interests in job matching. *Econometrica*, 52:47–57, 1984.

27 A. E. Roth. On the allocation of residents to rural hospitals: A general property of two-sided matching markets. *Econometrica*, 54(2):425–427, 1986.

28 Canadian Resident Matching Service. How the matching algorithm works. Web document available at `http://carms.ca/algorithm.htm`.

29 M. Sotomayor. Three remarks on the many-to-many stable matching problem. *Mathematical Social Sciences*, 38:55–70, 1999.

A A naive approach for finding max-size popular matchings

Given a many-to-many matching instance $G = (A \cup B, E)$, we investigate the possibility of constructing a corresponding one-to-one matching instance $G' = (A' \cup B', E')$ (with *strict* preference lists) in order to show a reduction from the max-size popular matching problem in G to one in G'. The vertex set A' will have $\mathsf{cap}(a)$ many copies $a_1, a_2, \ldots$ of every $a \in A$ and B' will have $\mathsf{cap}(b)$ many copies $b_1, b_2, \ldots$ of every $b \in B$; the edge set E' has $\mathsf{cap}(a) \cdot \mathsf{cap}(b)$ many copies of edge (a, b) in E. If $v \succ_u v'$ in G then we have $v_i \succ_{u_k} v'_j$ for each $i \in \{1, \ldots, \mathsf{cap}(v)\}$, $j \in \{1, \ldots, \mathsf{cap}(v')\}$, and $k \in \{1, \ldots, \mathsf{cap}(u)\}$. Among the copies $v_1, \ldots, v_{\mathsf{cap}(v)}$ of the same vertex v, we will set $v_1 \succ_{u_k} \cdots \succ_{u_k} v_{\mathsf{cap}(v)}$.

Given any matching $\tilde{M}$ in G', we define $\mathsf{proj}(\tilde{M})$ as the projection of $\tilde{M}$, which is obtained by dropping the subscripts of all vertices. We will now consider the *many-to-one* or the hospitals/ residents setting: so there are no multi-edges in $\mathsf{proj}(\tilde{M})$. As $\mathsf{proj}(\tilde{M})$ obeys all capacity bounds, it is a valid matching in G. It would be interesting to be able to show that every popular matching M in G has a *realization* $\tilde{M}$ in G' (i.e., $\mathsf{proj}(\tilde{M}) = M$) such that $\tilde{M}$ is a popular matching in G'.

However the above statement is not true as shown by the following example. Let $G = (R \cup H, E)$ where $R = \{p, q, r, s\}$ and $H = \{h, h', h''\}$ where $\mathsf{cap}(h) = 2$ and $\mathsf{cap}(u) = 1$ for all other vertices u. The preference lists are as follows:

p: h, h''	h: p, q, r, s
q: h, h'	h': q
r: h	h'': p
s: h	

Consider the matching $N = \{(p, h), (q, h'), (r, h)\}$. We show below that N satisfies the sufficient condition for popularity as given in Theorem 4. The proof of Claim 15 follows the same approach as used in the proof of Theorem 6.

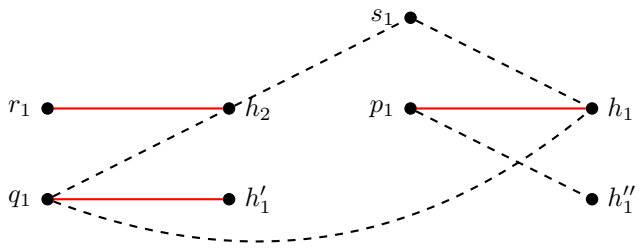


Figure 2 The edges of the matching N' are in red and the non-matching edges in G'_N are dashed. For simplicity, we have not included last resort neighbors here. Note that the edge (q_1, h_2) is a blocking edge to N' as both q_1 and h_2 prefer each other to their respective partners in N', i.e., $\mathsf{wt}_N(q_1, h_2) = 2$ and $\mathsf{wt}_N(e) = 0$ for all other edges e in G'_N.

▶ **Claim 15.** *N is popular in G.*

Proof. We have $R' = \{p_1, q_1, r_1, s_1\}$ and $H' = \{h_1, h_2, h_1', h_1''\}$. Here we use the notation introduced at the beginning of Section 3: let $N' = \{(p_1, h_1), (q_1, h_1'), (r_1, h_2)\}$ (see Fig. 2).

We need to show that every $(R' \cup H')$-complete matching in the weighted graph G'_N has weight at most 0. We will show this by constructing a *witness* or a solution to the dual LP corresponding to the primal LP which is the $(R' \cup H')$-complete max-weight matching problem in G'_N. This solution is the following: $\alpha_{p_1} = \alpha_{h_1} = \alpha_{s_1} = \alpha_{h_1''} = 0$ while $\alpha_{q_1} = \alpha_{h_2} = 1$ and $\alpha_{r_1} = \alpha_{h_1'} = -1$. The above solution is dual-feasible since every edge in G'_N is covered by the sum of α-values of its endpoints – in particular, note that $\alpha_{q_1} + \alpha_{h_2} = 2 = \mathsf{wt}_N(q_1, h_2)$. The dual optimal solution is at most $\sum_{u \in R' \cup H'} \alpha_u = 0$. So the primal optimal solution is also at most 0, in other words, N is a popular matching in G. ◀

Note that the graph G' has two *extra* edges relative to G'_N: these are (p_1, h_2) and (r_1, h_1). With respect to realizations of N in G', there are 2 candidates: these are $N_1 = \{(p_1, h_1), (q_1, h_1'), (r_1, h_2)\}$ and $N_2 = \{(p_1, h_2), (q_1, h_1'), (r_1, h_1)\}$.

▶ **Claim 16.** *Neither N_1 nor N_2 is popular in G'.*

Proof. Consider the matching $M_1 = \{(p_1, h_1''), (q_1, h_2), (r_1, h_1)\}$. The vertices p_1, h_1, and h_1' prefer N_1 to M_1 while the vertices q_1, h_2, r_1, and h_1'' prefer M_1 to N_1 and s_1 is indifferent. Thus M_1 is more popular than N_1, i.e., N_1 is not a popular matching in G'.

Consider the matching $M_2 = \{(p_1, h_1), (q_1, h_1'), (s_1, h_2)\}$. The vertices r_1 and h_2 prefer N_2 to M_2 while the vertices p_1, h_1, and s_1 prefer M_2 to N_2 and q_1, h_1', and h_1'' are indifferent. Thus M_2 is more popular than N_2, i.e., N_2 is not a popular matching in G'. ◀

Summarizing, there may exist popular matchings in G that cannot be realized as popular matchings in G'. Thus in order to claim that that $\mathsf{proj}(\tilde{M})$ is a *max-size* popular matching in G when $\tilde{M}$ is a max-size popular matching in G', it needs to be shown that there is at least one max-size popular matching in G that can be realized as a popular matching in G'.

Non-Adaptive Data Structure Bounds for Dynamic Predecessor Search

Joseph Boninger[1], Joshua Brody[2], and Owen Kephart[3]

1 **Swarthmore College, Swarthmore, PA, USA**
`jboning1@swarthmore.edu`
2 **Swarthmore College, Swarthmore, PA, USA**
`joshua.e.brody@gmail.com`
3 **Swarthmore College, Swarthmore, PA, USA**
`okephar1@swarthmore.edu`

Abstract

In this work, we continue the examination of the role *non-adaptivity* plays in maintaining dynamic data structures, initiated by Brody and Larsen. We consider non-adaptive data structures for predecessor search in the w-bit cell probe model. In this problem, the goal is to dynamically maintain a subset T of up to n elements from $\{1, \ldots, m\}$, while supporting insertions, deletions, and a predecessor query $\textsc{Pred}(x)$, which returns the largest element in T that is less than or equal to x. Predecessor search is one of the most well-studied data structure problems. For this problem, using non-adaptivity comes at a steep price. We provide exponential cell probe complexity separations between (i) adaptive and non-adaptive data structures and (ii) non-adaptive and memoryless data structures for predecessor search.

A classic data structure of van Emde Boas solves dynamic predecessor search in $O(\log \log m)$ probes; this data structure is adaptive. For dynamic data structures which make non-adaptive updates, we show the cell probe complexity is $O\left(\min\{\frac{\log m}{\log(w/\log m)}, \frac{n \log m}{w}\}\right)$. We also give a nearly-matching $\Omega\left(\min\{\frac{\log m}{\log w}, \frac{n \log m}{w \log w}\}\right)$ lower bound. We also give an $\Omega(m/w)$ lower bound for memoryless data structures.

Our lower bound technique is tailored to non-adaptive (as opposed to memoryless) updates and might be of independent interest.

1998 ACM Subject Classification E.1 Data Structures

Keywords and phrases dynamic data structures, lower bounds, predecessor search, non-adaptivity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.20

1 Introduction

The goal in a dynamic data structure problem is to maintain a database that changes over time, while supporting *queries* and *updates* to this data structure. A natural objective is to support both efficient queries and efficient updates. Often, either one is easily accomplished, but for many dynamic data structure problems, the optimal worst-case runtime on the maximal query/update time is polynomial. Nevertheless, proving such a data structure lower bound appears well beyond our current understanding. In fact, the highest lower bound for *any* dynamic data structure is currently $\Omega((\log n/\log \log n)^2)$ [8, 4, 17]. Identifying a dynamic data structure problem which supports a polynomial number of queries, which has a provably polynomial lower bound on either query or update time is one of the biggest open problems in data structures.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 20; pp. 20:1–20:12

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

Given the current difficulty in showing large data structure lower bounds, it is natural to ask if one can prove large lower bounds for restricted classes of data structures. Brody and Larsen [2] initiated a study of data structure bounds for *non-adaptive* data structures. In a non-adaptive data structure, the memory cells probed during a query or update are chosen in advance, independent of the contents of those cells. For many data structure problems, the optimal solution is indeed non-adaptive. Brody and Larsen [2] showed polynomial lower bounds for a large class of problems, both for when queries are non-adaptive and when updates are non-adaptive. It is worth noting that their lower bounds for non-adaptive updates were under an even more severe restriction called *memoryless updates*.

1.1 The Cell Probe Model

In the cell probe model defined by Yao [18], a data structure consists of a series of memory *cells* or *words*, each storing a w-bit integer. We assume there are at most 2^w cells in the data structure, so that each cell can be addressed using a single w-bit integer. When a query or update is executed, a number of cells are *probed*. During a query or update, which cell is probed next may depend arbitrarily on previous computation, and similarly what is written to a cell during an update may also depend arbitrarily on previous computation in the update. We define the *query complexity* of a data structure, denoted t_q, as the maximum number of cells probed during a query. Similarly, the *update complexity*, denoted t_u, is the maximum number of cells probed during an update. As mentioned previously, it is often easy to design data structures that have either low t_q or low t_u; our goal is to minimize the *cell probe complexity* of a data structure, defined as $\max\{t_q, t_u\}$.

Cell probe complexity measures only the number of memory accesses used by the data structure. All other computation is not counted and essentially given for free. In particular, no assumption is made about what CPU computations are allowed. The generality of the cell probe model makes it ideally suited for studying lower bounds; these lower bounds will apply to other computation models which make more CPU assumptions.

1.2 Non-Adaptive Data Structures

Given the current barriers in proving high cell probe lower bounds for dynamic data structures, it makes sense to study restricted classes of data structures. Non-adaptivity is a natural restriction. We examine several different kinds of non-adaptivity, listed below.

- **Non-Adaptive Query Algorithm.** A cell probe data structure has a non-adaptive query algorithm if the cells probed when answering a query depend only on the query itself, and not on the contents of previously probed cells.
- **Non-Adaptive Update Algorithm.** A cell probe data structure has a non-adaptive update algorithm if the cells probed when answering an update depend only on the update itself, and not on the contents of previously probed cells.
- **Memoryless Update Algorithm.** A cell probe data structure has a memoryless update algorithm if the update algorithm is non-adaptive, and additionally the contents written to a cell depend only on the current contents of the cell and the update itself, and not on contents of other cells previously probed during the update operation.

A *non-adaptive data structure* is a cell probe data structure that has non-adaptive queries and updates. Similarly, a *memoryless data structure* is a cell probe data structure that has non-adaptive queries and memoryless updates.

The study of non-adaptivity in data structures is of both theoretical and practical interest. Our original motivation in examining non-adaptive data structures was to generalize the class of non-adaptive structures for which we can prove lower bounds, with an eye towards proving polynomial lower bounds on dynamic data structures. This work expands the range of non-adaptive data structures for which we can prove strong lower bounds.

From a more practical perspective, there are many real-world computational settings where the number of memory accesses does not capture the true performance of a data structure or algorithm. Instead, the computational bottleneck is the *rounds of communication* between disk and memory (e.g. the external memory or I/O model [1, 16, 15]) or between different processors (distributed/parallel computation [6, 7, 13]) This is directly related to how adaptive the algorithm or data structures are. We anticipate that a better understanding of the role non-adaptivity plays in data structures will shed further insight into data structures for other models of computation as well.

1.3 Predecessor Search and Our Results

In this work, we primarily focus on two dynamic data structure problems: Predecessor and Max. In the Predecessor data structure problem, we are to maintain a dynamic set of up to n elements $T \subseteq [m]$[1], initially empty, with updates $\text{Insert}(i), \text{Delete}(i)$ which insert/delete i from T. Each query $\text{Pred}(i)$ should return the largest $x \in T$ that is less than or equal to i. In the Max problem, we again maintain a dynamic set $T \subseteq [m]$ with the same insert and delete operations. Additionally, there is a single query Max(), which must return the largest element in T.

In either problem, we assume that each element of the universe $[m]$ can fit into a single cell of memory; i.e., we assume that $w \geq \log m$. We make a further reasonable assumption that $w \leq m^{0.25}$.

Predecessor search is one of the most well-studied data structure problems, and it deserves a complete study. Moreover, it is a common component in other data structure problems such as binary search trees. In this way, our lower bounds are likely to apply to other data structure problems as well.

As mentioned previously, the van Emde Boas tree [14] is an adaptive data structure for Predecessor with cell probe complexity $O(\log \log m)$. Our main result shows that adaptivity is crucial for such an upper bound.

▶ **Theorem 1.** *Let* $\alpha := \min\{n, w/2\}$. *Then, any non-adaptive data structure solving* Predecessor *with* $t_u = O(\log m)$ *must satisfy*

$$t_q \geq \frac{\alpha \log m}{2w \log(w \cdot t_u)} .$$

▶ **Corollary 2.** *Fix any non-adaptive data structure for the dynamic predecessor problem with query time* t_q, *and update time* t_u. *If* $n \geq w/2$ *then we have*

$$\max(t_q, t_u) = \Omega\left(\frac{\log m}{\log w}\right) .$$

If $n < w/2$, *then we have*

$$\max(t_q, t_u) = \Omega\left(\frac{n \log m}{w \log w}\right) .$$

[1] We use $[m]$ to denote the set $\{1, \ldots, m\}$, and $[y, z]$ to denote the set of integers $\{y, y+1, \ldots, z\}$.

Is our non-adaptive lower bound the best possible? Our next result shows that it is close to optimal.

▶ **Theorem 3.** *There exists a non-adaptive data structure for* Predecessor *with* $t_q, t_u = O\left(\frac{\log m}{\log(w/\log m)}\right)$.

When $w = \Theta(\log m)$ and n is large, our non-adaptive data structure bounds are loose by a factor of $\log\log m$. When n is large and $w \geq (\log m)^{1+\Omega(1)}$, our bounds are tight. For smaller values of n, our non-adaptive data structure is suboptimal, as the trivial data structure which stores an array of up to n values and probes the entire data structure on each operation uses $n\log(m)/w$ words. In this case, our lower bound is off by a factor of $\log w$.

Either way, Theorem 1 can be seen to interpolate between our non-adaptive data structure from Theorem 3 and the trivial upper bound, allowing for a smooth tradeoff between n and m.

Finally, we give a very strong lower bound for memoryless data structures solving Max. Note that Max is essentially Predecessor with a further restriction that Pred(m) is the only query allowed, so this lower bound holds for Predecessor as well.

▶ **Theorem 4.** *Any memoryless data structure for* Max *must have* $\max\{t_q, t_u\} \geq m/w$.

This lower bound is much higher than the cell probe complexity of the first trivial solution and is a clean illustration of what is impossible to do with memoryless data structures.

1.4 Previous Results

The study of dynamic data structures, and data structures for predecessor search, has a long history; here we give a brief synopsis. Yao [18] introduced the cell probe model and gave cell probe bounds for the membership problem. Fredman and Saks [5] created the chronogram technique and showed $\Omega(\log(n)/\log\log(n))$ bounds for several dynamic data structure problems, including partial sums. This remained the highest lower bound for any dynamic data structures problem until Pătraşcu and Demaine gave an $\Omega(\log n)$ bound for dynamic connectivity. Pătraşcu and Thorup [10, 11] gave strong deterministic and randomized lower bounds for static predecessor search. Pătraşcu also developed an exciting line of attack for dynamic data structure lower bounds by connecting several conjectured hard problems to a communication problem called Multiphase [9]. Pătraşcu conjectured a strong communication lower bound for Multiphase and showed that this lower bound implies polynomial lower bounds for several dynamic data structures. Chattopadhyay et al. [3] disproved the strongest of Pătraşcu's conjectures for Multiphase, but not in a way which invalidates the implication for polynomial data structure lower bounds.

The highest lower bounds to date for any dynamic data structure problem is the bound $\Omega((\log(n)/\log\log(n))^2)$ of Larsen [8] for dynamic range counting. A similar lower bound was later given by Clifford et al. [4] for Matrix Vector Multiplication and Pătraşcu's Multiphase problem.

Brody and Larsen [2] initiated the study of non-adaptive bounds for dynamic data structures and showed polynomial lower bounds for a number of problems including Multiphase. The techniques in Chattopadhyay et al. [3], which preceded [2], give the same lower bounds. Neither of these works showed lower bounds for non-adaptive (but not memoryless) updates.

▶ Remark. Recently Ramamoorthy and Rao [12] independently proved non-adaptive data structure bounds for Predecessor Search. Ramamoorthy and Rao consider a similar set of secondary problems (median, minimum vs. maximum) and obtain similar bounds, showing

for predecessor search that either $t_q \geq \frac{\log m}{\log w + \log\log m}$ or $t_u \geq \Omega\left(\frac{t_q m^{1/(2t_q+2)}}{\log m}\right)$. Our bounds are stronger when $t_q = \Omega(\frac{\log m}{\log\log m})$. Additionally, our analysis and bounds provide tradeoffs between the universe size m, word size w, and maximum number of items in the set n; Ramamoorthy and Rao consider only the first two parameters. In both works, the lower bounds for predecessor search apply even when insertions are the only updates. Ramamoorthy and Rao only require queries to be non-adaptive; we require both queries and insertions to be non-adaptive.

Both papers have been developed independently and in parallel.

1.5 Our Technique

For the non-adaptive data structure lower bound (Theorem 1), we prove something stronger than is stated in the theorem. Specifically, we show that there must be a large set of cells C along with a set $A \subseteq [m]$ that is reasonably large such that for each $i \in A$, Pred(i) queries every cell in C. We build this set C iteratively by using the pigeonhole principle along with an encoding argument.

For complete details, see Section 2; we give a high-level sketch here. We begin with $C := \emptyset$ and $A = [m]$, and consider the first update Insert(1). This update has the potential to affect every query. Furthermore, for any operation, the set of cells probed are fixed and chosen in advance. Therefore, for each $i \in A$, the cells probed by Pred(i) and by Insert(1) must intersect. Insert(1) probes at most t_u cells, so by the pigeonhole principle, there must be some cell c that is probed by Insert(1) and by at least m/t_u queries. We fix that cell and set $A := \{i : \text{Pred}(i) \text{ probes } c\}$.

Next, we claim that if C is not too large and A is not too small, then there must be some $j \in A$ and a not-too-small fraction of A such that Insert(j) and Pred(i) intersect *outside of* C. This claim, formalized in Lemma 9, is nontrivial and is our main technical tool. We prove this using an encoding argument—essentially, we show that if there was no such update, then C would contain at least $\approx \log(m)^2$ bits of information, contradicting the assumption that $|C|$ is small. From here we apply another pigeonhole argument to show that there must be some cell outside of c that is probed by a large enough fraction of the remaining queries. Alternating applications of the pigeonhole argument and our main technical lemma allows us to grow C iteratively, up to a size of $|C| \geq \frac{\alpha \log m}{2w \log(w \cdot t_u)}$.

Theorems 3 and 4 use more standard techniques. Our $O(\log m)$ non-adaptive data structure uses range trees, and our $\Omega(m/w)$ lower bound on memoryless data structures uses a direct encoding argument—we're able to use a memoryless data structure for Max to encode an arbitrary subset of $[m]$, requiring m/w words.

Before getting into the lower bound proofs, we summarize the encoding arguments common to data structure lower bounds.[2]

1.5.1 The Coding Lower Bound

In a typical encoding argument, an encoder is tasked with communicating information (say, an element $x \in \mathcal{S}$ from a finite set) to a decoder. The encoder can encode this information in any number of ways, as long as the decoder can unambiguously recover the information. In

[2] Encoding arguments are often phrased in terms of input distributions and Shannon entropy. In this work, we focus on deterministic data structure bounds, and simplify the Coding Lower Bound accordingly.

order for the decoder to recover the encoder's input, the encoder must at a minimum send a different message for each distinct input.

When applying encoding arguments to show data structure lower bounds, the encoder uses a data structure to encode an arbitrary element from $\mathcal{S}$. If the data structure was unreasonably efficient, then the length of the encoding would be too short for the decoder to recover the input without error. We conclude that the data structure cannot be *too efficient*. This intuition is formalized in the definition and fact below.

▶ **Definition 5** (Encoding Procedure)**.** An *encoding procedure* for a finite set $\mathcal{S}$ is a pair of functions $\text{ENC} : \mathcal{S} \to \{0,1\}^k, \text{DEC} : \{0,1\}^k \to \mathcal{S}$ such that for all $x \in \mathcal{S}$, we have

$$\text{DEC}(\text{ENC}(x)) = x \ .$$

The *length* of the encoding is k.

The key feature of an encoding procedure is that the encoder must send a different message for each element of $\mathcal{S}$. Otherwise, the decoder cannot decode without error.

▶ **Fact 6.** *[Coding Lower Bound]*
In any encoding procedure for a finite set $\mathcal{S}$, the length of the encoding must satisfy $k \geq \log(|\mathcal{S}|)$.

1.6 Roadmap

We prove Theorems 1, 3, and 4 in Sections 2, 3, and 4 respectively, and introduce notation relevant for each theorem in the relevant sections.

2 Non-Adaptive Lower Bound for Predecessor

For our non-adaptive lower bound, it is helpful to work with a more symmetric "wrap-around" variation of the standard Predecessor problem. In this variation, we define Pred(i) to be equal to

1. the largest $x \leq i$ in T, if such an element exists,
2. the largest $x \in T$, if T is nonempty but contains no elements $\leq i$, or
3. $\perp$ if T is empty.

It is easy to see that this variation affects the cell probe complexity by at most a factor of 2. We resist notation expansion and in this section use Predecessor to denote this wrap-around variation. The symmetry of this version of Predecessor will be useful because each update has the potential to affect any query.

For this lower bound, we will need to compare the sets of cells probed by different updates and queries. It will be helpful to introduce some notation to make this argument easier to express.

For any $i, j \in [m]$, we let u_j and q_i denote Insert(j) and Pred(i) respectively. By convention, we use a subscript j to refer to updates and i to refer to queries. We use U_j and Q_i to denote the set of cells probed by u_j, q_i respectively. We'll also abuse notation a bit and use $A \subseteq [m]$ to denote both a subset of indices and the corresponding subset of queries or updates.

▶ **Theorem 7.** *[Restatement of Theorem 1] Let* $\alpha := \min\{n, w/2\}$*. Any non-adaptive data structure solving dynamic predecessor with update time* $t_u = O(\log m)$*, and query time* t_q *must satisfy*

$$t_q \geq \frac{\alpha \log m}{2w \log(w \cdot t_u)} \ .$$

▶ **Corollary 8.** *Fix any non-adaptive data structure for the dynamic predecessor problem with query time* t_q*, and update time* t_u*. If* $n \geq w/2$ *then we have*

$$\max(t_q, t_u) = \Omega\left(\frac{\log m}{\log w}\right) \ .$$

If $n < w/2$*, then we have*

$$\max(t_q, t_u) = \Omega\left(\frac{n \log m}{w \log w}\right) \ .$$

The proof of Theorem 1 depends on the following technical lemma, whose proof we defer to Subsection 2.1.

▶ **Lemma 9** (Main Technical Lemma)**.** *Let* C *be a set of cells in the data structure, and let* $A \subseteq [m]$*. If*
1. $|A| \geq \sqrt{m}$*,*
2. $|C| \leq \frac{\alpha \log m}{5w}$*, and*
3. *for all* $i \in A$*,* q_i *probes all cells in* C*,*

then there exists $j \in A$ *and a subset* $A' \subseteq A$ *such that* $|A'| \geq \frac{|A|}{w^2}$ *and for each* $i \in A'$ *there is a cell* $c \notin C$ *such that* u_j *and* q_i *both probe* c*.*

At a high level, this lemma says that if we have a large enough set of queries A and a small enough set of cells C such that each query in A probes each cell in C, then there must be an update u_j that has a nontrivial intersection *outside of* C with a large subset of A.

Proof of Theorem 1. We prove this theorem by induction. Fix an arbitrary non-adaptive data structure for Predecessor. As mentioned in the introduction, we'll prove this theorem by iteratively growing a large set of cells C in the data structure and a not-too-small set of queries A such that each query in A probes each cell in C. If we can grow the set of cells until $|C| = \frac{\alpha \log m}{2w \log(w \cdot t_u)}$ while keeping the set of queries nonempty, the theorem will follow.

This intuition is captured by the following inductive claim.

▶ **Claim 10.** *For all integers* $1 \leq k \leq \frac{\alpha \log m}{2w \log(w \cdot t_u)}$*, there is a set of* k *cells* C *and a set queries* $A \subseteq [m]$ *such that*
1. $|A| \geq \frac{m}{w^{2(k-1)} t_u^k}$
2. $C \subseteq Q_i$ *for all* $i \in A$*.*

Setting $k = \frac{\alpha \log m}{2w \log(w \cdot t_u)}$ proves the theorem.

It remains to prove the claim. First, we prove the base case of $k = 1$. Fix an arbitrary update u_j, and note that U_j must intersect Q_i for each $i \in [m]$. Otherwise, the contents of the cells queried by q_i would be the same for the empty set and for $T = \{j\}$, but $\text{Pred}(i) = \perp$ when the set is empty, and $\text{Pred}(i) = j$ when the set is $\{j\}$. Note also that $|U_j| \leq t_u$, so by the pigeonhole principle, there must be a cell $c \in U_j$ probed by at least m/t_u queries $i \in [m]$. Fix this cell c, define $C := \{c\}$, and let A be the set of queries that probe c. This set of cells C and queries A fit the premise of Claim 10, completing the base case.

For the induction hypothesis, assume Claim 10 holds for some arbitrary $k < \frac{\alpha \log m}{2w \log(w \cdot t_u)}$.

In the induction step, we'll show that Claim 10 holds for $k+1$ as well. By the induction hypothesis, there is a set of k cells C_k and queries A_k such that $|A_k| \geq m/(w^{2(k-1)} t_u^k)$ and $C_k \subseteq Q_i$ for all $i \in A$. To invoke Lemma 9, $|A_k|$ must be at least $\sqrt{m}$. This holds as long as $k \lesssim \frac{\log(m)}{2\log(w^2 t_u)}$, which is valid since $\alpha \leq w/2$.

By Lemma 9, there is an update $j \in A_k$ and subset $A'_k \subset A_k$ such that $|A'_k| \geq |A_k|/w^2$ and for each $i \in A'_k$ there is a cell $c \notin C_k$ such that u_j and q_i both probe c. Next, we again use the pigeonhole principle. Since $|U_j \setminus C| \leq |U_j| \leq t_u$, there must be a cell $c \in U_j \setminus C$ and a set $A''_k \subseteq A'_k$ such that $|A''_k| \geq |A'_k|/t_u$ and such that for each $i \in A''_k$, Q_i probes c. Set $C_{k+1} := C \cup \{c\}$ and $A_{k+1} := |A''_k|$. Note that $|A_{k+1}| \geq |A_k|/w^2 t_u$ and that $C_{k+1} \subseteq Q_i$ for all $i \in A_{k+1}$. The sets C_{k+1}, A_{k+1} fit the premise of Claim 10 for $k+1$, completing the induction step. ◀

2.1 Proof of Main Technical Lemma

We prove Lemma 9 using an encoding argument—we show that if the lemma is false, then we can use C to encode more than $|C| \cdot w$ bits of information, a contradiction.

Before delving into the technical details of the proof, we introduce some notation. Say that a set of cells C *satisfies* (u_j, q_i) if $U_j \cap Q_i \subseteq C$; that is, if C contains all cells probed by both u_j and q_i. Similarly, for a set $T \subseteq [m]$, say that C satisfies (T, q_i) if C satisfies (u_j, q_i) for all $j \in T$. Lemma 9 states that there is $j \in A$ and a large subset $A' \subseteq A$ (with $|A'| \geq |A|/w^2$) such that for all $i \in A'$, C *fails* to satisfy (u_j, q_i).

Proof of Lemma 9. Towards a contradiction, assume that for all $j \in A$, there are less than $|A|/w^2$ queries $i \in A$ such that the given set of cells C fails to satisfy (u_j, q_i). We'll then use the data structure and C to encode the following set:

$$\mathcal{S} := \{T \subseteq A : |T| = \alpha \text{ and } |j - j'| \geq \frac{|A|}{w} \text{ for all } j, j' \in T\} .$$

$\mathcal{S}$ is the set of all possible "spread-out" subsets of A with size α.

▶ **Claim 11.** $|\mathcal{S}| \geq 2^{\frac{\alpha \log(m)}{4}}$.

Proof. We construct a subset of $\mathcal{S}$ with the desired size. Let $x_1, \ldots, x_\alpha$ be arbitrary elements of $\{1, \ldots, |A|/w\}$. Set $y_i := \frac{(2i-1)|A|}{w} + x_i$, and set $T := \{y_i\}$. Note that $y_1 > \frac{|A|}{w}$, $y_\alpha \leq \frac{(2\alpha-1)|A|}{w} + \frac{|A|}{w} = \frac{2\alpha|A|}{w} \leq |A|$, and that by definition of T we have

$$\frac{2i-1}{w}|A| < y_i \leq \frac{2i}{w}|A| = \frac{2(i+1)-1}{w}|A| - \frac{|A|}{w} \leq y_{i+1} - \frac{|A|}{w} .$$

This means that $y_{i+1} - y_i \geq \frac{|A|}{w}$ for all i, hence T is a valid element of $\mathcal{S}$. There are $\frac{|A|}{w}$ choices for each x_i, and α elements of T, so there are $(|A|/w)^\alpha$ choices for T. Thus, we have

$$|\mathcal{S}| \geq \left(\frac{|A|}{w}\right)^\alpha = 2^{\alpha \log(|A|/w)} \geq 2^{\frac{\alpha}{4}\log(m)} ,$$

where the final inequality holds because $w \leq m^{1/4}$ and $|A| \geq \sqrt{m}$. ◀

Encoding Procedure. Given an arbitrary $T \in \mathcal{S}$, the encoder takes the non-adaptive data structure, initially storing an empty set. She then inserts each $j \in T$. After performing all insertions, the encoder sends the contents of each cell in C.

Decoding Procedure. The decoder first takes the non-adaptive data structure, initialized to store the empty set. Then, she overwrites the contents of each cell in C using the encoder's message. The decoder then executes q_i for each $i \in A$ and outputs the set of all elements that appear at least $\frac{|A|}{2w}$ times as answers; that is, the decoder returns the set $T' := \{j \in A : \text{ there are at least } \frac{|A|}{2w} \text{ elements } i \text{ with Query(i)} == j\}$.

Analysis. It is easy to see that the length of the encoding is $w \cdot |C| \leq \frac{\alpha \log(m)}{5}$ bits, since the encoder sends the memory contents of each cell in C. Next, we claim that the decoder correctly recovers T. By assumption, we have that for all $j \in A$, the set of cells C satisfies (u_j, q_i) for all but at most $\frac{|A|}{w^2}$ queries. Therefore, for any $T \in \mathcal{S}$, C satisfies (T, q_i) for all but at most $\frac{|A|}{w^2}\alpha < \frac{|A|}{2w}$ queries $i \in A$.

Now, consider what happens when C satisfies (T, q_i). For any $j \in T$, C contains all cells probed by both u_j and q_i. Since this holds for all $j \in T$, C contains all cells that changed during insertions *that were probed by* q_i. Thus the decoder can correctly compute q_i when C satisfies (T, q_i).

When C does not satisfy (T, q_i), then the decoder is not guaranteed to correctly compute q_i; we assume without loss of generality that this is an error. The decoder executes query q_i for each $i \in A$, but computes this query incorrectly whenever C does not satisfy (T, q_i). Moreover, since the decoder does not know T in advance, she cannot know a priori which queries failed. We claim that because less than $\frac{|A|}{2w}$ queries are not satisfied, the decoder still has enough information to recover T.

To see this, take any $j \in T$. By construction, $|j - j'| \geq \frac{|A|}{w}$ for any $j, j' \in T$. Hence j is the correct answer to query q_i for all $i \in [j, j + \frac{|A|}{w} - 1]$. Even if all errors were in this range, there would still be more than $\frac{|A|}{2w}$ queries for which the decoder correctly computes j. Hence, the decoder will place $j \in T'$. Conversely, consider any $j \notin T$. Then, j is not a correct answer for any query. In the worst case, the decoder computes j for each possible query on which she errs. Since there are less than $\frac{|A|}{2w}$ such queries, the decoder will not place $j \in T'$. The decoder adds j to T' if and only if $j \in T$, hence the decoder correctly outputs T.

We've shown how to encode an arbitrary $T \in \mathcal{S}$ using $w \cdot |C|$ bits. By Fact 6 and Claim 11, we must have

$$w \cdot |C| \geq \log(|\mathcal{S}|) \geq \frac{\alpha \log m}{4} .$$

Therefore, we must have $|C| \geq \frac{\alpha \log(m)}{4w}$, contradicting our assumption that $|C| \leq \frac{\alpha \log m}{5w}$. ◀

3 Non-Adaptive Upper Bound

In this section, we give an $O(\frac{\log m}{\log(w/\log m)})$ non-adaptive upper bound for Predecessor by using a form of range tree.

Proof of Theorem 3. We first handle the case where $w = \lceil \log m \rceil$, so each cell stores a single element from the universe $[m]$. Then, we adjust the construction to handle $w \gg \log m$.

Let k be the least integer such that $2^k \geq m$. Our data structure consists of a complete binary tree with 2^k leaves, labeled $1, \ldots, 2^k$. At each node v in the tree, we store the largest i such that (i) $i \in T$ and (ii) i is a descendant of v. If no such i exists, we store $\perp$. Note that each leaf i stores either i or $\perp$, and that the root node stores the maximal element of T. Additionally, an interior node v with children l, r stores the maximum of what is contained in the cells of l, r, treating $\perp$ as 0. In other words, $\max(v) := \max\{\max(l), \max(r)\}$.

To execute Insert(i), for each node v on the path from i to the root (including leaf i), the data structure checks to see if i is now the largest element among descendants of v and updates appropriately if so. Note that the set of nodes probed corresponds to all nodes on the path from leaf i to the root. This is fixed in advance, so Insert(i) is indeed a non-adaptive update.[3]

Implementing Delete(i) is similar. Using the invariant that a the cell corresponding to node v maintains the max of whatever is stored in its children, the data structure must query both children of node v. This must happen for each node v on the path from leaf i up to the root, resulting in twice as many cell probes as an insert. However, as with insertions, *which* cells to probe are known in advance, so the data structure remains non-adaptive. Unlike insertions, these updates are not memoryless, since updating node v depends on the deletion, the current contents of the cell, and the contents of both children.

To implement Pred(i), we traverse the path from the root down to leaf i. Each time we take the right child, the data structure queries the cell corresponding to the left child. We also query the node corresponding to leaf i, and return the maximal element found, or $\perp$ if all queried cells returned $\perp$. In this way, the range $\{1, \ldots, i\}$ is partitioned into a series of subranges, with at most one subrange per level of the binary tree. This sketch describes the query algorithm as walking down the tree, but the nodes that are queried depend only on i itself, and so they can be again chosen in advance.

Insertions, deletions, and queries can all be performed non-adaptively by querying at most two cells per level of the tree. The tree has size $2^k < 2 \cdot m$, so the height is at most $k = O(\log m)$. Since each operation probes at most two cells per level of the range tree, the query complexity is also $O(\log m)$.

Finally, suppose that $w \gg \log m$. In this case, we can pack $w/\log m$ elements into a single word. Fix h such that $2^h = w/\log m$, so that $h = \log(w/\log m)$. We modify the original range tree argument to pack subtrees of height h into a single cell. So, for example, one cell stores the root value and all values of nodes less than h away from the root. For each node at level h of the range tree, we store in a single cell all descendants at distance less than h from this node, and so on. Insertions, deletions, and queries happen as before, but the w-bit memory cell containing the value stored at each node is probed as opposed to the node itself. We still probe at most 2 cells per level, but this time, our "cells" consume h levels of the original range tree. As a result, our new query complexity is $O\left(\frac{\log m}{h}\right) = O\left(\frac{\log m}{\log(w/\log m)}\right)$. ◀

4 Memoryless Lower Bound for Predecessor

Our final result is a strong lower bound for the cell probe complexity of memoryless data structures that solve Predecessor. In fact, our result is a lower bound for a simpler problem Max. Max easily reduces to Predecessor, so the lower bound applies to both problems.

Proof of Theorem 4. The proof is a simple encoding argument. Let the encoder be given a set $T \subseteq [m]$, and let $\mathcal{D}$ be a memoryless data structure for Max.

The encoder encodes T by first preprocessing a copy of $\mathcal{D}$ and then inserting each element in T one at a time into $\mathcal{D}$. She then writes the contents of each cell probed by Max. Call these cells C_{Max}. Because the query algorithm is non-adaptive, C_{Max} can be determined without knowledge of their contents and will not change regardless of any updates that occur.

[3] In fact, insertions in this data structure are memoryless, since each cell update depends only on the insertion and the current contents of the cell.

The decoding protocol is as follows. The decoder preprocesses her own copy of $\mathcal{D}$ and initializes $T' = \emptyset$. Then, she writes to the cells in C_{Max} using the encoding provided by the encoder. The decoder then performs the following until $k = 0$:

1. Run Max() on $\mathcal{D}$. Let k be the value returned by Max().
2. If $k = \perp$, the decoder ends the decoding algorithm and outputs T'.
3. If $k > 0$, the decoder adds k to T', emulates Delete(k) on $\mathcal{D}$, and repeats the process.

Note that the decoder cannot completely execute the Delete(k) operations, but does not need to. Since queries and updates are non-adaptive, she knows which cells get probed by Delete(k). Additionally, since the updates are memoryless, the contents of each cell written by Delete(k) are a function of Delete(k) and the current contents of the cell. The decoder only needs to maintain the cells probed by Max(). Since she is given the initial contents of these cells by the encoder, and since she knows which update operations to perform, she can maintain the contents of the cells probed by Max(). The decoder might not be able to maintain cells outside of C_{Max} that are probed by updates, but she does not need to, since these cells are not queried by Max(). By repeating this process as long as Max returns nonzero elements, the decoder can recover all of T.

We now analyze the length of the encoding to determine a lower bound on t_q. The encoder sends w bits for each cell in C_{Max}. Since $|C_{\text{Max}}| = t_q$ by definition, the length of the encoding is wt_q. The encoding is for an arbitrary subset $S \subseteq [m]$, so by the coding lower bound, any encoding must be at least m bits long. Thus, we get $wt_q \geq m$, hence $t_q \geq m/w$. ◀

References

1 Alok Aggarwal, Jeffrey Vitter, et al. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

2 Joshua Brody and Kasper Green Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. *Theory of Computing*, 11:471–489, 2015. doi:10.4086/toc.2015.v011a019.

3 Arkadev Chattopadhyay, Jeff Edmonds, Faith Ellen, and Toniann Pitassi. Upper and lower bounds on the power of advice. *SIAM Journal on Computing*, 45(4):1412–1432, 2016.

4 Raphael Clifford, Allan Grønlund, and Kasper Green Larsen. New unconditional hardness results for dynamic and online problems. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1089–1107. IEEE, 2015.

5 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM ACM Symposium on Theory of Computing (STOC)*, pages 345–354. ACM, 1989.

6 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522. ACM, 2010.

7 Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *ACM SIGACT News*, 42(1):82–96, 2011.

8 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proc. 44th ACM Symposium on Theory of Computing (STOC)*, pages 85–94, 2012.

9 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, pages 603–610, 2010.

10 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

11 Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 555–564, 2007.

12 Sivaramakrishnan Natarajan Ramamoorthy and Anup Rao. Non-adaptive data structure lower bounds for median and predecessor search from sunflowers. https://eccc.weizmann.ac.il/report/2017/040/, 2017.

13 Nir Shavit. Data structures in the multicore age. *Communications of the ACM*, 54(3):76–84, 2011.

14 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 75–84, 1975.

15 Elad Verbin and Qin Zhang. The limits of buffering: a tight lower bound for dynamic membership in the external memory model. *SIAM Journal on Computing*, 42(1):212–229, 2013.

16 Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, 33(2):209–271, 2001.

17 Omri Weinstein and Huacheng Yu. Amortized dynamic cell-probe lower bounds from four-party communication. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 305–314. IEEE, 2016.

18 Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 28(3):615–628, 1981.

On Colourability of Polygon Visibility Graphs*

Onur Çağırıcı[1], Petr Hliněný[2], and Bodhayan Roy[3]

1 **Faculty of Informatics, Masaryk University Brno, Czech Republic**
onur@mail.muni.cz
2 **Faculty of Informatics, Masaryk University Brno, Czech Republic**
hlineny@fi.muni.cz
3 **Faculty of Informatics, Masaryk University Brno, Czech Republic**
b.roy@fi.muni.cz

Abstract

We study the problem of colouring the visibility graphs of polygons. In particular, we provide a polynomial algorithm for 4-colouring of the polygon visibility graphs, and prove that the 6-colourability question is already NP-complete for them.

1998 ACM Subject Classification G.2.2 Graph algorithms, F.2.2 Geometrical problems and computations

Keywords and phrases polygon visibility graph, graph coloring, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.21

1 Introduction

Visibility graphs are widely studied graph classes in computational geometry. Geometric sets such as sets of points or line segments, polygons, polygons with obstacles, etc., all can correspond to specific visibility graphs, and have uses in robotics, signal processing, security paradigms, decomposing shapes into clusters [1, 2, 7, 11, 15]. Here we study the visibility graphs of simple polygons in the Euclidean plane and henceforth in the paper, all polygons are simple unless stated otherwise.

Given an n-vertex polygon P (not necessarily convex) in the plane, two points p and q of P are said to be *mutually visible* if, and only if the line segment $\overline{pq}$ does not intersect the exterior of P. The n-vertex visibility graph $G(V, E)$ of P is defined as follows. The vertex set V of G contains a vertex v_i if, and only if, the polygon P contains the point p_i as its vertex. The edge set E of G contains an edge $\{v_i, v_j\}$ if, and only if, the points p_i and p_j are mutually visible. Given a polygon P in the plane, we can compute its visibility graph G in $\mathcal{O}(n^2)$ time using the polygon triangulation method [8, 17]. Hence, in this paper, we slightly abuse notation by not distinguishing between a polygon P and its visibility graph G and referring to a polygon vertex p_i as to the corresponding G-vertex v_i.

Visibility graphs of polygons have been studied with respect to various theoretical and practical computational problems. The complexities of several popular optimization problems have been determined for visibility graphs of polygons. A geometric variation of the dominating set problem, namely polygon guarding, is one of the most studied problems in computational geometry and is known as the Art Gallery Problem [15]. It has been studied extensively for both polygons with and without holes and has been found to be NP-hard in both cases [12, 16]. Besides, given a polygon, computing a maximum independent set is

* P. Hliněný and O. Çağırıcı are supported by the Czech Science Foundation, project no. 17-00837S.

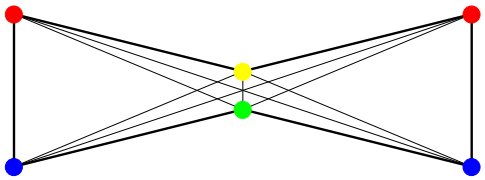


Figure 1 A visibility graph that is without a K_5, non planar but is 4-colourable.

known to be hard, due to Shermer [20], while computing a maximum clique has been shown to be in polynomial time by Ghosh et al. [19].

A *proper vertex colouring* of a graph is an assignment of labels or colours to the vertices of the graph so that no two adjacent vertices have the same colours. Henceforth, when we say colouring a graph, we refer to proper vertex colouring. The *chromatic number* of a graph is defined as the minimum number of colours used in any proper colouring of the graph. Visibility graph colouring has been studied for various types of visibility graphs. Babbitt et al. gave upper bounds for the chromatic numbers of k-visibility graphs of arcs and segments [3]. Kára et al. characterized 3-colourable visibility graphs of point sets and described a super-polynomial lower bound on the chromatic number with respect to the clique number of visibility graphs of point sets [10]. Pfender showed that, as for general graphs, the chromatic number of visibility graphs of point sets is also not upper-bounded by their clique numbers [18]. Diwan and Roy showed that for visibility graphs of point sets, the 5-colouring problem is NP-hard, but 4-colouring is solvable in polynomial time [5].

The problem of *colouring* the visibility graphs of given polygons has been studied in the special context where each internal point of the polygon is seen by a vertex, whose colour appears exactly once among the vertices visible to that point [4,6,9]. However, little is known on colouring visibility graphs of polygons without such constraints. Although 3-colouring is NP-hard for general graphs [14], in particular it is rather trivial to solve it for visibility graphs of polygons in polynomial time using a greedy approach. Already with 4 colours the same question has been open so far.

In this paper we settle (nearly in full) the complexity question of the general problem of colouring polygonal visibility graphs, which was declared open in 1995 by Lin and Skiena [13]. We provide a polynomial-time algorithm to find a 4-colouring of the visibility graph of a given polygon, if such a colouring exists. On the other hand, we provide a reduction showing that the question of k-colourability of the visibility graph of a given simple polygon is NP-complete for any $k \geq 6$. Only the problem of 5-colourability is left open.

2 4-Colouring visibility graphs

In this section, we study the algorithmic question of 4-colourability of the visibility graph of a given polygon. The full structure of 4-colourable visibility graphs is not yet known and it seems to be non-trivial. For instance, if a visibility graph is planar, it is obviously 4-colourable. Though, if such a graph contains K_5, then it is neither planar nor 4-colourable, but a visibility graph not containing any K_5 may be non-planar yet 4-colourable (Figure 1).

The related algorithmic problem of 3-colouring visibility graphs is rather easy to resolve as follows. Every simple polygon can be triangulated and, in such a triangulation, every non-boundary edge is contained in two triangles. One can then proceed greedily edge by edge: Suppose a triangle has already been coloured, and it shares an edge with a triangle that is not fully coloured. Then the two end vertices of the shared edge uniquely determine the colour of the third vertex of the uncoloured triangle.

Our algorithm essentially generalizes the 3-colouring method for 4-colouring. We first divide the polygon into *reduced polygons*. A polygon P is called a reduced polygon, if every chord of P is intersected by another chord of P. After the division, we find and colour a triangle (a K_3 subgraph) with three distinct colours in each reduced subpolygon. Subsequently, whenever we find an uncoloured vertex v adjacent to some three vertices coloured with three distinct colours (such as, to an already coloured triangle), we can uniquely colour also v, by the fourth colour. We will show that we can exhaust all vertices of a reduced subpolygon in this manner. Furthermore, we check for possible colouring conflicts – since the colouring process is unique, this suffices to solve 4-colourability.

Altogether, this will lead to the following theorem.

▶ **Theorem 1.** *The 4-colourability problem is decidable in polynomial time for visibility graphs of simple polygons, and if a 4-colouring exists, then it can be computed in polynomial time.*

In the coming proof, consider a polygon P and its visibility graph $G(V, E)$, embedded on P. Hereafter we slightly abuse notation by equating P and G. Since we want to 4-colour P, we assume that G has no K_5 (or we answer 'no'). We denote the clockwise polygonal chain of P from a vertex u to a vertex v as $\Gamma(u, v)$.

One can easily see that it is enough to focus on reduced P in our proofs. Indeed, assume an edge uv of G which is a chord of P and not crossed by any other chord. We can partition P into subpolygons P_1 and P_2, where $P_1 = (u\,\Gamma(u, v)\,v)$ and $P_2 = (v\,\Gamma(v, u)\,u)$. Since no edge of G has one end in $P_1 \setminus P_2$ and the other in $P_2 \setminus P_1$, the polygons P_1 and P_2 can be 4-coloured separately and merged again (provided that P is 4-colourable).

Let u and v be two vertices of P. The *shortest path* between u and v is a (graph) path from u to v in G such that the sum of the Euclidean lengths of its edges is minimized. Such a shortest path between u and v is unique in P and is denoted as $\Pi(u, v)$. Observe that all non-terminal vertices of a shortest path are non-convex [7]. We will assume an implicit ordering of vertices on $\Pi(u, v)$ from u to v. When we say that some vertex w is the first (or last) vertex on $\Pi(u, v)$ with a certain property, we mean that w precedes (respectively, succeeds) all other vertices with that property on $\Pi(u, v)$.

For a proof of Theorem 1, we have got the following sequence of claims. Consider, in all of them, a K_5-free reduced polygon P and its three vertices t_1, t_2, t_3 forming a triangle $T \subseteq G$. Assume that T is already coloured (which is unique up to a permutation of the colours). Suppose that v_i is an uncoloured vertex, such that an edge incident to v_i intersects T. Then we have the following lemmas.

▶ **Lemma 2.** *Assume that two vertices $v_i \in \Gamma(t_1, t_2)$ and $v_j \in \Gamma(t_2, t_3)$ see each other, and the edge v_iv_j intersects t_1t_2 and t_2t_3. Then the colours of all vertices on the four paths $\Pi(t_1, v_i)$, $\Pi(t_2, v_i)$, $\Pi(t_2, v_j)$ and $\Pi(t_3, v_j)$, including v_i, v_j themselves, are uniquely determined by the colours of T.*

Proof. We prove the claim by induction on the four paths. As the base case, the first vertices of these paths are the vertices of T, which are already assigned different colours.

For the induction step, assume that $\Pi(t_1, v_i)$, $\Pi(t_2, v_i)$, $\Pi(t_2, v_j)$ and $\Pi(t_3, v_j)$ have been coloured till vertices v_a, v_b, v_c and v_d respectively. Also, their immediate uncoloured successors on $\Pi(t_1, v_i)$, $\Pi(t_2, v_i)$, $\Pi(t_2, v_j)$ and $\Pi(t_3, v_j)$ are v_p, v_q, v_r and v_s respectively. We aim to show that the colours of at least one of v_p, v_q, v_r and v_s is uniquely determined by the already coloured vertices.

If v_p does not see v_b and any predecessor of v_b on $\Pi(t_2, v_i)$, then v_q must see v_a or some predecessor of v_a on $\Pi(t_1, v_i)$. We have the following cases.

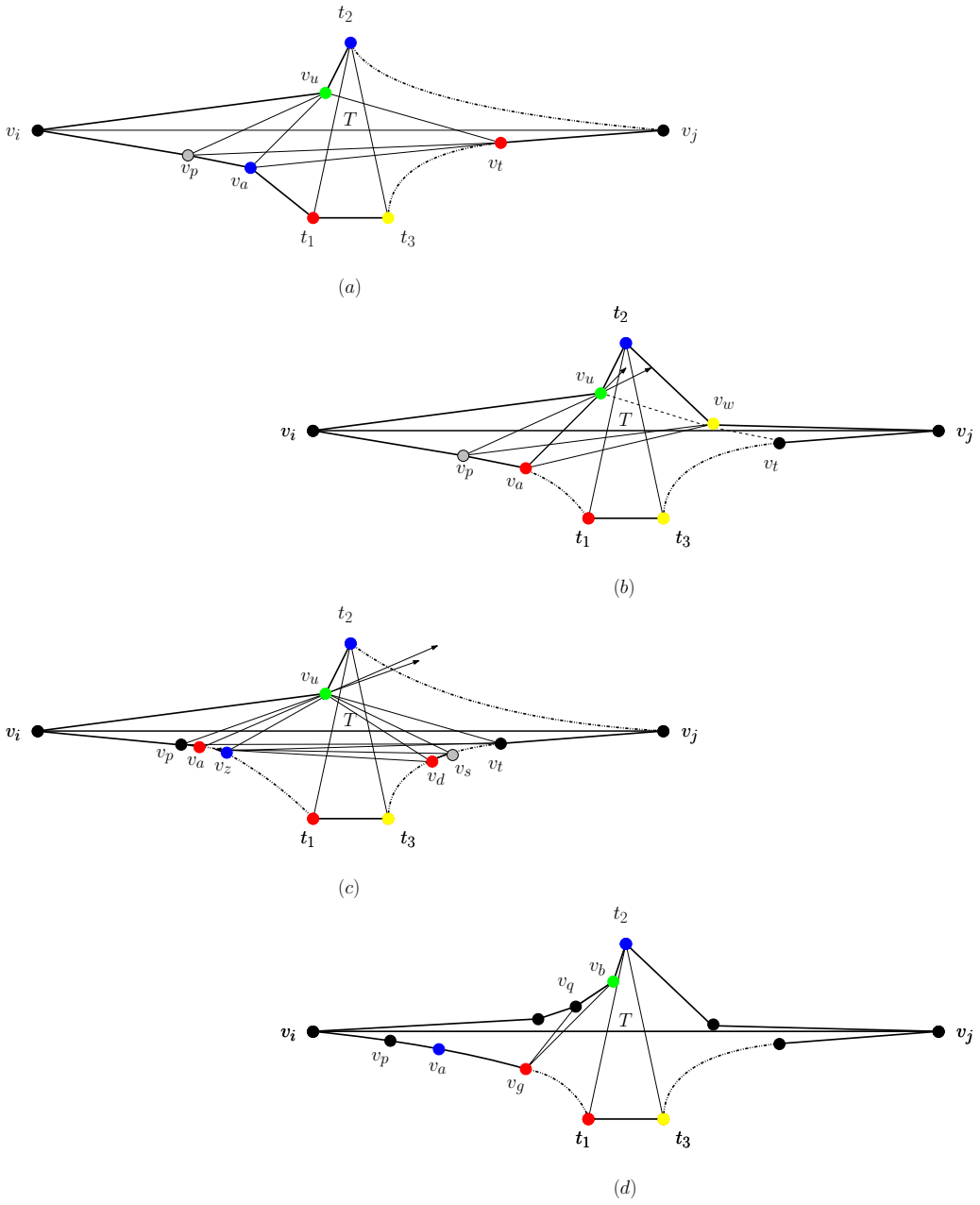


Figure 2 Illustration of the proof of Lemma 2: The vertices whose colours shall be uniquely determined next, are now drawn in gray. Polygonal boundaries containing multiple vertices not included in the figure are drawn with dashed lines. (a) v_p forms a K_4 with v_a, v_t and v_u. (b) v_p forms a K_4 with v_a, v_u and v_w. (c) v_s forms a K_4 with v_u, v_d and v_z. (d) v_g, v_q and v_b form a K_3.

Case 1: v_p sees v_b or some predecessor of $v_b$on $\Pi(t_2, v_i)$.

By definition, v_p is the immediate successor of v_a on $\Pi(t_1, v_i)$, so v_p must see v_a. The right tangent of v_a to $\Pi(t_2, v_i)$ lies to the right of the right tangent of v_p to $\Pi(t_2, v_i)$. So, if the right tangent of v_p to $\Pi(t_2, v_i)$ touches $\Pi(t_2, v_i)$ at a point v_u, then v_a sees v_u. Note that either $v_u = v_b$ or v_u precedes v_b on $\Pi(t_2, v_i)$. In any case, v_u is already coloured. Since v_p,

v_a and $\Pi(t_3, v_j)$ lie on the same side of v_iv_j, and v_p is nearer to v_iv_j than v_a is, v_p and v_a see a vertex v_t of $\Pi(t_3, v_j)$. If v_u also sees v_t, and v_t is already coloured, then the claim is proved (Figure 2(a)). So we consider the other two cases, namely, that either v_u does not see v_t, or v_t is not yet coloured.

Subcase 1.a: v_u does not see v_t. Since v_t and v_u lie on different sides of v_iv_j, some vertex of $\Pi(t_2, v_j)$ must be blocking v_u and v_t. Let v_w be the first vertex of $\Pi(t_2, v_j)$ blocking v_u and v_t. Then v_u sees v_w. The vertex v_w is closer to v_iv_j than v_u is. Also, v_w lies to the right of $\overrightarrow{v_av_u}$ and $\overrightarrow{v_pv_u}$, and to the left of $\overrightarrow{v_av_t}$ and $\overrightarrow{v_pv_t}$. Then the only possible blockers between v_w and v_p or v_a can be from $\Pi(t_2, v_i)$. But all the vertices on $\Pi(t_2, v_i)$ preceding v_u are further from v_iv_j than v_u is. So, there can be no such blocker, and v_w must be visible from both v_a and v_p. If v_w is already coloured, then the claim is proved (Figure 2(b)). Suppose that v_w is not already coloured. Then consider v_r, which precedes v_w on $\Pi(t_2, v_j)$. The vertices v_r and v_c are consecutive on $\Pi(t_2, v_j)$ and hence see each other. Since $\Pi(t_2, v_j)$ and $\Pi(t_1, v_i)$ are on opposite sides of v_iv_j, the vertices v_c and v_r either see v_a or a vertex preceding v_a on $\Pi(t_1, v_i)$. Let v_x be the last coloured vertex of $\Pi(t_1, v_i)$ seen by both v_c and v_r. If $v_x \neq v_a$ then let v_y be the last vertex of $\Pi(t_2, v_i)$ that blocks v_c from the successor of v_y on $\Pi(t_1, v_i)$. Then v_y must be visible from v_x, v_r and v_c. Since v_x precedes v_a on $\Pi(t_1, v_i)$, and v_y precedes v_b on $\Pi(t_2, v_i)$, both v_x and v_y must be already coloured. So, T uniquely determines the colour of v_r. If $v_x = v_a$ then since v_u is on the right tangent of v_a to $\Pi(t_2, v_i)$, both v_c and v_r see v_u. Hence, T uniquely determines the colour of v_r. Now we move to the second subcase.

Subcase 1.b: v_u sees v_t, but v_t is not yet coloured. Since v_u sees v_t, $\Pi(t_2, v_j)$ is a concave chain and the edge t_1t_3 exists in P, v_u must see every predecessor of v_t on $\Pi(t_2, v_j)$. This means that both v_d and v_s see v_u. The vertex v_s must see the vertex (say, v_y) of $\Pi(t_1, v_i)$ where the right tangent from v_d touches $\Pi(t_1, v_i)$, because the last vertices v_i and v_j of concave chains $\Pi(t_1, v_i)$ and $\Pi(t_3, v_j)$ see each other. Also, the left tangent of v_u to $\Pi(t_1, v_i)$ must touch $\Pi(t_1, v_i)$ at a vertex equal to or preceding v_y. Thus, all three of v_s, v_d and v_u see a common vertex v_z on $\Pi(t_1, v_i)$ which precedes v_a, since v_u and v_t see v_a. Thus, v_z is already coloured, and v_u, v_d and v_z form a K_4 with v_s and uniquely determine the colour of v_s (Figure 2(c)).

Case 2: v_p does not see v_b or some predecessor of v_b on $\Pi(t_2, v_i)$.

Here, we have the opposite situation to Case 1. Then v_q sees v_a or some predecessor of v_a on $\Pi(t_1, v_i)$. Let the left tangents from v_q and v_b touch $\Pi(t_2, v_i)$ at v_e and v_f respectively. Either v_b sees v_e or v_q must be a blocker between v_b and v_e. In this case, v_b and v_q see the last vertex v_g of $\Pi(t_1, v_i)$ that is not blocked from v_b by v_q. Since v_b does not see v_p, the vertex v_g must be already coloured (Figure 2(d)). Then, again, v_b and v_q see a vertex v_t on $\Pi(t_2, v_j)$. Now, some already coloured vertex v_u in $\Pi(t_2, v_i)$, adjacent to v_b and v_q might also see v_t, which may or may not be coloured. Or else, v_t might be blocked from such a vertex v_u by a vertex v_w of $\Pi(t_3, v_j)$. It can be seen that each of these arguments can be augmented similar to the subcases of Case 1, a K_4 can be found and the colour of one of the vertices v_p, v_q, v_r and v_s can be uniquely determined. ◀

▶ **Corollary 3.** *If any vertex v_i of P sees a vertex of T and their visibility edge crosses one of the edges of T, then the colour of v_i is uniquely determined by the colours of T.*

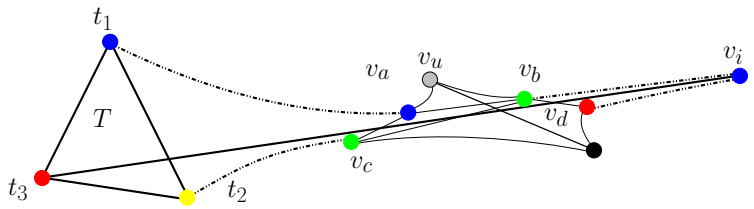


Figure 3 The vertex v_a has an edge incident to one of the vertices of v_a, v_b, v_c, where v_a, v_b and v_c lie on the already coloured shortest paths from t_1 and t_3 to v_i.

Proof. Without loss of generality, suppose that v_i sees t_1, and v_it_1 crosses t_2t_3. Then $v_j = t_1$, $\Pi(t_2, v_j) = t_2t_1$ and $\Pi(t_1, v_j) = t_1$, and Lemma 2 proves the claim. ◀

▶ **Lemma 4.** *If a reduced polygon is 4-colourable, then it has a unique 4-colouring (up to permutation of colours).*

Proof. Consider a triangle T in a reduced polygon P. If P is not just T, then at least one edge of T is not a boundary edge of P. Without loss of generality, let t_1t_2 be such an edge. Since P is reduced, there must be a vertex on $\Gamma(t_1, t_2)$ such that an edge incident to v_i crosses t_1t_2. By Lemma 2 and Corollary 3, if P is 4-colourable, then all vertices on the paths $\Pi(t_1, v_i)$ and $\Pi(t_2, v_i)$, including v_i have a 4-colouring uniquely determined by T. In case t_2t_3 or t_3t_1 are not boundary edges of P, we can similarly find v_j on $\Gamma(t_2, t_3)$ and v_k on $\Gamma(t_3, t_1)$ and uniquely 4-colour $\Pi(t_2, v_j)$, $\Pi(t_3, v_j)$, $\Pi(t_3, v_k)$ and $\Pi(t_1, v_k)$. Now, all the remaining uncoloured vertices of P are on polygonal chains of the form $\Gamma(v_a, v_b)$, where v_a and v_b are two consecutive vertices in one of the six paths mentioned above. Furthermore, no vertex in the polygonal chain $\Gamma(v_a, v_b)$, other than v_a and v_b, is coloured. Without loss of generality, let v_a and v_b be two consecutive vertices on $\Pi(t_1, t_2)$. If v_av_b is not a boundary edge of P, then since P is reduced, there must be an uncoloured vertex v_u in $\Gamma(v_a, v_b)$ such that an edge incident to v_u crosses v_av_b. This edge is either incident to a vertex of $\Pi(t_2, v_i)$, or crosses an edge of $\Pi(t_2, v_i)$. Consider the case where such an edge to a vertex of $\Pi(t_2, v_i)$ exists. Then consider a vertex v_w that is closest to v_av_b among all the vertices of $\Pi(t_2, v_i)$ that see a vertex (say, v_z) of $\Gamma(v_a, v_b)$. Since the edge v_wv_z exists, v_w cannot be blocked by any vertex of $\Pi(t_1, v_i)$. Due to the choice of v_w, no vertex of $\Pi(t_2, v_i)$ can block v_w from v_a or v_b. So, v_w sees both v_a and v_b. Now consider the case where no vertex of $\Gamma(v_a, v_b)$ sees any vertex of $\Pi(t_2, v_i)$, but some vertex of $\Gamma(v_a, v_b)$ sees some vertex of $\Gamma(v_c, v_d)$, where v_c and v_d are consecutive points on $\Pi(t_2, v_i)$. Without loss of generality, assume that v_c precedes v_d in $\Pi(t_2, v_i)$. Then v_c must see both v_a and v_b (Figure 3), for otherwise a vertex of $\Gamma(v_a, v_b)$ must have an edge with some vertex of $\Pi(t_2, v_i)$ acting as a blocker for v_c, contrary to our assumption. Then, in the above two cases, based on the triangle $v_av_bv_w$ and $v_av_bv_c$, respectively, again Lemma 2 and Corollary 3 can be used to uniquely determine a 4-colouring for $\Pi(v_a, v_u)$ and $\Pi(v_b, v_u)$.

Now we generalize the above procedure. Let $T_0 = \{T\}$, and $S_0 = \{\Pi(t_1, v_i), \Pi(t_2, v_i), \Pi(t_2, v_j), \Pi(t_3, v_j), \Pi(t_3, v_k), \Pi(t_1, v_k)\}$. Note that we have assumed that none of the edges of T are boundary edges. If some edges of T are boundary edges then S_0 will have less elements. By the above procedure, we can uniquely 4-colour all the vertices of all elements of S_0. Now, all the uncoloured vertices lie on $\Gamma(v_a, v_b)$, where v_a and v_b are consecutive vertices of some element of S_0. For each such v_av_b, we find a new triangle $v_av_bv_c$ or $v_av_bv_d$, and two new shortest paths of the form $\Pi(v_a, v_u)$ and $\Pi(v_b, v_u)$. Let T_1 denote the set of all such new triangles, and S_1 denote the set of all new shortest paths obtained from T_0 and S_0. Now,

the remaining uncoloured vertices must line on polygonal chains of the form $\Gamma(v_e, v_f)$ where v_e and v_f are two consecutive vertices of some element of S_1. In general, following the same method we can always construct T_{i+1} and S_{i+1} from T_i and S_i, until all vertices of P are coloured. Since in each step, the colours of vertices are uniquely determined, it follows that if P has a 4-colouring, then it must be unique. ◀

Our algorithm to decide 4-colourablity of visibility graphs of polygons is given next.

Algorithm 1: Algorithm to decide 4-colourablity of visibility graphs of polygons

Input: A simple polygon P with the visibility edges
Output: If P is 4-colourable or not. If so, then proper 4-colouring of P.
Decompose P into reduced subpolygons $P_1, \ldots, P_k$;
foreach *reduced subpolygon* P_i **do**
  Locate a triangle;
  repeat
    Compute a 4-colouring for vertices on the polygonal chain of each non-boundary edge of the triangle;
    `/* Using the method of Lemma 2 and Corollary 3 */`
    Continue the process using the method of Lemma 4;
  until *Each vertex in* P_i *is coloured*;
end
if *two adjacent vertices receive the same colour* **then**
  Output 'non-4-colourable';
  Terminate;
end
Rejoin the reduced subpolygons such that each pair P_i, P_j of subpolygons having a common edge have exactly two vertices in common;
Permute the colours of the vertices so that there is no conflict.

Now, in light of the above Algorithm 1, we prove Theorem 1.

Proof of Theorem 1: Lemma 2 and Corollary 3 colour the shortest path from a triangle to a vertex uniquely with 4 colours. Lemma 4 repeats the process exhausting all vertices. Since the colour of each vertex is uniquely determined by some three previously coloured vertices, the resulting 4-colouring is unique, if P is 4-colourable. Consequently, if a conflict is found, it follows that P is not 4-colourable. So, the algorithm is correct.

Let the number of vertices and edges in G be n and m respectively. The chords that do not cross any other chord, can be found in $O(m^2)$ time. Thus, the decomposition of P into reduced subpolygons takes $O(m^2)$ time. Shortest paths from a triangle to a vertex can be found in $O(n)$ time. While computing the colouring on the shortest paths, a pointer can be kept on each of the shortest paths, and the colouring takes $O(n)$ time. The colouring step can be iterated at most once for each vertex, so the complexity for all vertices is $O(n^2)$. Checking for conflict takes $O(m)$ time. Finally, rejoining the reduced subpolygons takes $O(n)$ time. Thus, the complexity of the algorithm is $O(m^2)$. ◀

3 Hardness of 6-colourability

In this section we prove that the problem of deciding whether the visibility graph G of a given simple polygon P can be properly coloured with 6 colours, is NP-complete.

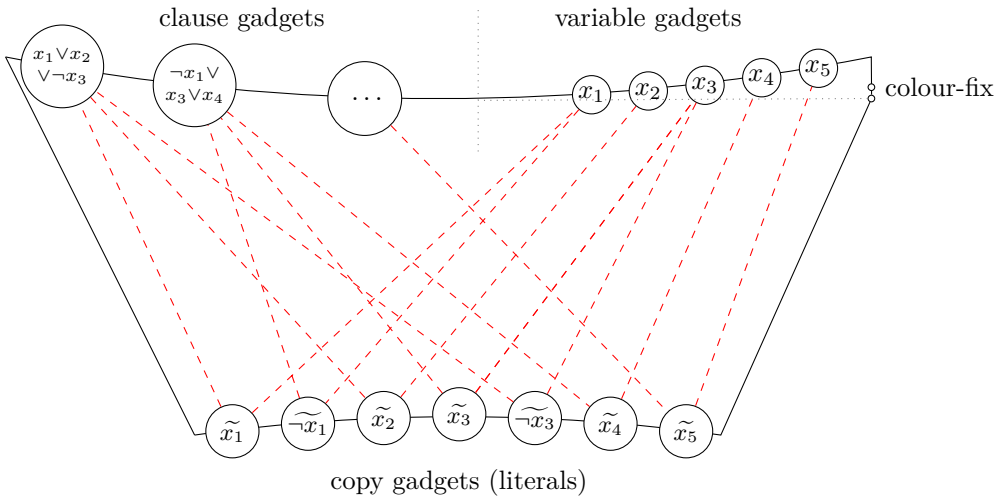


Figure 4 A scheme of the polygon P constructed from a 3-SAT formula in Section 3. Note that the top and bottom part are placed on slightly concave arcs, which block undesired visibilities between gadgets. The colour-fix gadget is placed so that it can see none of the clause gadgets. The red dashed lines show "visibility communication" between related variable and copy gadgets (the literals), and between related copy and clause gadgets.

Membership of our problem in NP is trivial (since G can be efficiently computed from P and then a colouring checked on G). We are going to present a polynomial reduction from the NP-hard problem of *Not-all-equal* 3-*SAT*: Given is a formula Φ in the conjunctive normal form, such that every clause of Φ contains exactly 3 literals, and the task is to find a (satisfying) assignment to the variables of Φ such that every clause contains at least one true and at least one false literal. For that we will construct a polygon P such that its proper 6-colourings correspond to satisfying assignments of Φ. We start with a rough informal outline of the construction.

- Our polygon P will consist of one *colour-fixing* gadget, a series of *variable* gadgets (one per each variable of Φ), a series of *copy* gadgets (one per each literal occurring in Φ), and a series of *clause* gadgets (one per each clause of Φ). Visibility edges will allow "communication" between variable gadgets and their corresponding copies representing the literals, and between the literals and their clause gadgets. Apart from that, there will be no other visibility relation between "internal" vertices of our gadgets. See Figure 4.
- Assume that the visibility graph G of P can be properly 6-coloured. The role of the colour-fixing gadget is to fix these six colours so that precisely two of them, named here as *red* and *blue*, can be used to colour the vertices representing the variables of Φ. The remaining four colours play an auxiliary role; they are used to colour those vertices which "separate" the gadgets from each other, or to "moderate" clause gadgets. More specifically, *yellow* and *orange* colour separating vertices at the variable and clause side ("top"), and *light* and *dark green* colour separating vertices of the copy gadgets ("bottom").
- For each variable x_i of Φ, there will be a variable gadget $R(x_i)$ which, in particular, contains two mutually visible internal vertices named x_i and $\neg x_i$. They must hence be coloured red and blue, or blue and red, encoding the logical value of x_i in Φ. There is no direct influence between colouring decisions of distinct variable gadgets.

- For each literal ℓ occurring in Φ (such as $\ell = x_j$ or $\ell = \neg x_j$ for some variable x_j), there will be a copy gadget $P(\ell)$ which, in particular, contains an internal vertex named $\widetilde{\ell}$. Visibility between the gadgets $R(x_j)$ and $P(\ell)$ are adjusted so that $\widetilde{\ell}$ must receive, in any 6-colouring of G, the same colour as that of ℓ in $P(\ell)$. Furthermore, the point $\widetilde{\ell}$ is positioned so that it is visible only from selected vertices of the corresponding clause gadget of ℓ, as specified later (note that different literals of x_j have separate copy gadgets).
- For each clause $c = \ell_1 \vee \ell_2 \vee \ell_3$ of Φ, there will be a clause gadget $S(c)$ whose vertices can selectively see, among all internal vertices of all the copy gadgets, exactly the points $\widetilde{\ell_1}$, $\widetilde{\ell_2}$, $\widetilde{\ell_3}$. This selected visibility is such that, locally, the clause gadget $S(c)$ can be properly coloured iff not all three points $\widetilde{\ell_1}$, $\widetilde{\ell_2}$, $\widetilde{\ell_3}$ have the same colour. Furthermore, for any satisfying assignment of Φ, proper colourings of all the clause gadgets can be properly combined together.

Note also that, within the presented reduction scheme, 6 colours is a necessary minimum. We need two colours for the separating vertices of the top part, another two such at the bottom part, and then two more colours are required to encode logical values of the variables.

Altogether, this will lead to the following:

▶ **Theorem 5.** *The problem – given a simple polygon P in the plane, to decide whether the visibility graph of P is properly k-colourable – is NP-complete for every $k \geq 6$.*

Proof. As mentioned, the problem is in NP since one can construct the visibility graph G of P in polynomial time [8,17] and then verify a colouring. In the opposite direction, we reduce from the NP-complete Not-all-equal 3-SAT problem. Given a 3-SAT formula Φ, we efficiently construct a polygon P such that the visibility graph G of P is k-colourable if, and only if, Φ is not-all-equal satisfiable. In the proof, we refer to the previous construction outline.

We construct only the least case $k = 6$ since for higher k the construction can be easily adjusted (simply saying, we can add more shades of green to the copy gadgets), as detailed at the end. As for our terminology, a *gadget* is a consecutive part of the polygonal chain of P. The vertices of each gadget are divided to *internal* and *external* ones (except clause gadgets which have no external vertices). The internal vertices define the function of each gadget, while the external ones serve as separators from the neighbouring gadgets. Two consecutive gadgets may share their external vertices.

The unique *colour-fix gadget* A is a (convex) chain of 6 vertices $a_1, \ldots, a_6$ in this clockwise order (cf. Figure 4) which see each other. Without loss of generality, in every 6-colouring of G the colours of external vertices a_1, a_2 are *yellow and orange*, the colours of internal a_3, a_4 are *red and blue* and the colours of external a_5, a_6 are *light and dark green*.

For each variable x_i of Φ, there is one *variable gadget* $R(x_i)$ formed as a convex chain of 4 vertices $r_1^i, r_2^i, r_3^i, r_4^i$, hence seeing each other (Figure 5 top). Furthermore, the external vertices r_1^i, r_4^i of $R(x_i)$ are visible from all four a_3, a_4, a_5, a_6 of the colour-fix gadget A, while the internal vertices r_2^i, r_3^i of $R(x_i)$ are visible from a_5, a_6 of A. Consequently, r_1^i, r_4^i can only be coloured yellow and orange, and r_2^i, r_3^i can only receive colours red and blue. The internal vertices r_2^i and r_3^i are nicknamed x_i and $\neg x_i$, respectively, and their colours will represent their value 'true' (red) and 'false' (blue). Together, the variable gadgets $R(x_i)$, $i = 1, 2, \ldots, n$, are chained together (the order does not really matter) such that r_4^i of $R(x_i)$ is identified with r_1^{i+1} of $R(x_{i+1})$, and the rightmost r_4^n is identified with a_1 of the colour-fix gadget A. Globally, the variable gadgets are arranged in a nearly-straight concave position (cf. Figure 4), so that they do not see each other (except consecutive ones at the shared external vertices). The points a_1, a_2 of A are also part of this concave arrangement.

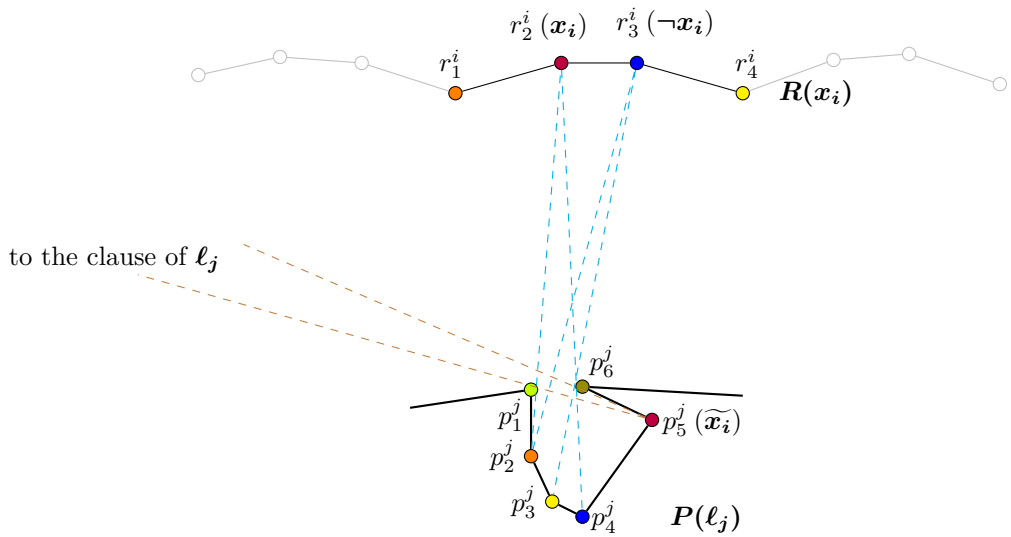


Figure 5 Detailed arrangement of the copy gadget $P(\ell_j)$ of a literal $\ell = x_i$, where the cyan dashed lines show important visibility relations between $P(\ell_j)$ and the variable gadget $R(x_i)$. Specifically, all 6 available colours have to be used on $P(\ell_j)$ and, thanks to p_4^j not seeing r_3^i, the only viable choice is to colour p_4^j same as r_3^i (blue) and then p_5^j to get the same colour as r_2^i (red). Brown dashed lines show the visibility angle of p_5^j which will be used by the corresponding clause gadget.

For each literal ℓ_j occurring in Φ (such as $\ell_j = x_i$ or $\ell_j = \neg x_i$ for some variable x_i), there is one separate *copy gadget* $P(\ell_j)$ formed as a convex chain of 6 vertices $p_1^j, \ldots, p_6^j$ (shaped as a "cavity"). Among them, p_1^j and p_6^j are the external ones, visible in particular from all four a_1, a_2, a_3, a_4 of the colour-fix gadget A. The remaining internal vertices $p_2^j, p_3^j, p_4^j, p_5^j$ are visible only from selected vertices of the variable gadget $R(x_i)$ and of the clause gadget corresponding to the literal ℓ_j. Specifically, the arrangement is as depicted in Figure 5 (for the case $\ell_j = x_i$): r_2^i sees all points of $P(\ell_j)$ except p_5^j and r_3^i sees all except p_4^j, p_5^j, while r_1^i cannot see p_2^j and r_4^i cannot see p_3^j. Besides this, r_1^i may possibly see p_4^j and r_4^i may see p_2^j, but this does not matter. The special point p_5^j (previously named $\widetilde{\ell_j}$) will see, except $P(\ell_j)$, only a later specified part of a clause gadget to which ℓ_j belongs to. The purpose of this arrangement is to force p_5^j to the same colour as r_2^i has (Claim 6), while keeping full flexibility of selecting the visibility angle of p_5^j.

If, on the other hand, the considered literal is $\ell_j = \neg x_i$, we only slightly shift the points in Figure 5, such that r_2^i could not see p_2^j and r_3^i would see p_4^j. (Alternatively, we could avoid this case by considering Φ without negations, in which case not-all-equal satisfiability remains hard.) Globally, all the copy gadgets $P(\ell_j)$ are chained together (the order does not matter) again in a nearly-straight concave shape as in Figure 4, but this time without identification of their external vertices. In particular, p_6^j is a neighbour of p_1^{j+1} on the polygonal chain of P but, importantly, p_6^j cannot see p_6^{j+1}. The points a_5, a_6 of the colour-fix gadget A are also part of this concave arrangement.

Then, for each clause $c_m = (\ell_{m_1} \vee \ell_{m_2} \vee \ell_{m_3})$ of Φ, there is one *clause gadget* $S(c_m)$ formed as a nearly-straight convex chain of 4 vertices $s_1^m, s_2^m, s_3^m, s_4^m$. All points of $S(c_m)$ are visible from a_5, a_6 of the colour-fix gadget A, and so all four remaining colours (including red and blue) have to be used on $S(c_m)$. Furthermore, the point $p_5^{m_1}$ of the copy gadget $P(\ell_{m_1})$

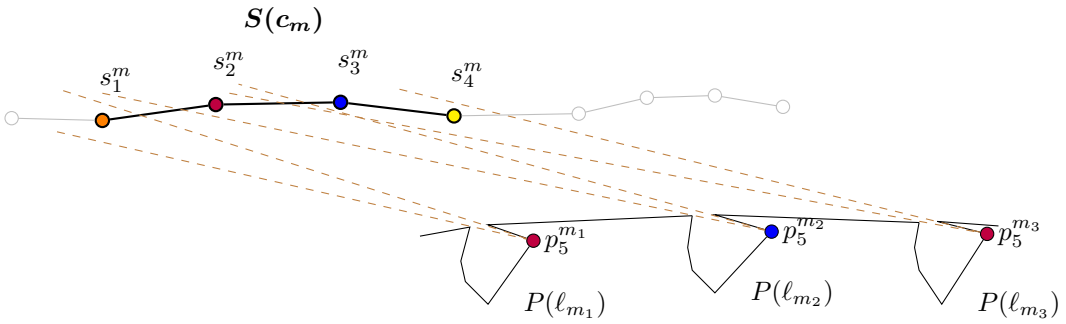


Figure 6 Detailed arrangement of the gadget $S(c_m)$ of a clause $c_m = (\ell_{m_1} \vee \ell_{m_2} \vee \ell_{m_3})$, where dashed brown lines delimit the visible angles of the points $p_5^{m_1}, p_5^{m_2}, p_5^{m_3}$. Since all clause gadgets see also colours light and dark green (e.g., of A), $S(c_m)$ can be properly coloured by itself iff not all of $p_5^{m_1}, p_5^{m_2}, p_5^{m_3}$ come with the same colour (which mimics not-all-equal satisfiability). The copy gadgets of one clause do not have to be consecutive, even though the picture shows them such.

sees exactly the point s_1^m, and likewise $p_5^{m_2}$ of $P(\ell_{m_2})$ sees exactly s_2^m. The point $p_5^{m_3}$ of $P(\ell_{m_3})$ sees both s_3^m, s_4^m. See Figure 6. The aim of this arrangement is that there would not be enough colours for whole $S(c_m)$ if all three $p_5^{m_1}, p_5^{m_2}, p_5^{m_3}$ came with the same colour. All the clause gadgets $S(c_m)$ are globally chained together, without vertex identification, in the same nearly-straight concave arrangement with the variable gadgets (cf. Figure 4). Although, no point of clause gadgets is visible from a_1, a_2, a_3, a_4 of the colour-fix gadget A.

Finally, one extra vertex (the bottom-left corner in Figure 4) is used to close the polygon P between the clause and copy sections. Validity of Theorem 5 is established from the following sequence of simple claims.

First assume that the visibility graph G of P is properly 6-coloured.

▶ **Claim 6.** *For every variable x_i of Φ, the vertices r_2^i and r_3^i of $R(x_i)$ receive colours blue and red, in either order. For every literal ℓ_j of Φ such that $\ell_j = x_i$ ($\ell_j = \neg x_i$, respectively), the vertex p_5^j of $P(\ell_j)$ receives the same colour as r_2^i (as r_3^i).*

Since (mutually visible) points r_1^i and r_4^i of $R(x_i)$ are visible from all four a_3, a_4, a_5, a_6 of the colour-fix gadget A, they must receive the colours of a_1, a_2 (yellow and orange). Then, since r_2^i and r_3^i are visible from a_5, a_6 and also from r_1^i, r_4^i, they must be coloured the same as a_3, a_4, which is red and blue.

For $\ell_j = x_i$, the points of $P(\ell_j)$ must be coloured as follows: p_1^j, p_6^j see a_1, a_2, a_3, a_4 of A, and so they have the same colours as a_5, a_6 (light and dark green). Furthermore, p_2^j, p_3^j are visible from r_2^i, r_3^i, and so they can be neither red nor blue. Consequently, p_4^j and p_5^j are red and blue (as r_2^i, r_3^i), and since r_2^i sees p_4^j, the only proper choice is to have p_5^j coloured the same as r_2^i. For $\ell_j = \neg x_i$, identical arguments lead to p_5^j being coloured the same as r_3^i. Claim 6 is finished.

▶ **Claim 7.** *Interpreting the colours of vertices p_5^j of literal ℓ_j as logical 'true' (red) and 'false' (blue), every clause of Φ receives at least one true and one false literal. Consequently, Φ is not-all-equal satisfiable.*

This claim is trivial; consider a clause $c_m = (\ell_{m_1} \vee \ell_{m_2} \vee \ell_{m_3})$. All points of $S(c_m)$ see a_5, a_6 of A, and so only the remaining four colours (yellow, orange, red, blue) are available for the four mutually visible vertices of the clause gadget $S(c_m)$. If all three points $p_5^{m_1}, p_5^{m_2}, p_5^{m_3}$ had the same colour (either red or blue), then the remaining three colours would not be enough for $S(c_m)$, and so both colours occur among $p_5^{m_1}, p_5^{m_2}, p_5^{m_3}$, as desired.

In the remaining direction of Theorem 5, we need to argue as follows.

▶ **Claim 8.** *The construction of P can be realized in a grid of polynomial size in $|\Phi|$. Consequently, the construction is a polynomial reduction.*

We refer to the sketch of P in Figure 4. Both the top and bottom concave chains can be realized as "fat" parabolas, requiring only rough resolution of $\mathcal{O}(|\Phi|^2)$. We place all the gadgets (roughly) equally spaced along, with their external vertices on these parabolas. Positioning of all the vertices of the variable and clause gadgets, and of the colour-fix gadget, is natural and easy, requiring no finer resolution. The only delicate part is to precisely place the points of the copy gadgets. The external vertices p_1^j, p_6^j get placed very close to each other on the bottom parabola, and the internal ones are then fine-positioned so that they have the required visible angles (with respect to the upper parabola). This, for each copy gadget, is done independently of all other copy gadgets, and only an additional polynomial (cubic) sub-resolution is needed for the whole copy section. This finishes Claim 8.

▶ **Claim 9.** *If Φ is not-all-equal satisfiable, then G can be properly 6-coloured.*

We describe a desired proper 6-colouring of G of P. First, we colour the gadget A and the external vertices of the variable and copy gadgets. We give vertices $a_1, a_2, a_3, a_4, a_5, a_6$ of A colours yellow, orange, red, blue, dark green, light green in this order. In every copy gadget $P(\ell_j)$, we colour p_6^j dark green and p_1^j light green (as in Figure 5). The extra vertex of P added to the left of the copy section, gets colour dark green. Thanks to concave arrangement of the copy section, this is so far a proper partial colouring of G.

For variable gadgets $R(x_i)$, we alternate colouring of the external vertices – while r_1^i may be orange and r_4^i yellow, for the next one it is $r_1^{i+1} = r_4^i$ yellow and r_4^{i+1} orange, and so on, until the last $r_4^n = a_1$ is yellow. Again, thanks to concave arrangement of the variable section, this is so far a proper partial colouring of G.

Next, we assume a not-all-equal satisfying assignment of Φ. For a variable x_i, we colour r_2^i red and r_3^i blue if x_i is 'true', and we colour r_2^i blue and r_3^i red if x_i is 'false'. Then we correspondingly colour each copy gadget featuring x_i, as in Figure 5 – this is always possible since p_2^j may inherit the colour of r_1^i and p_3^j that of r_4^i and p_4^j that of r_3^i. So, p_5^j has he same colour as r_2^i (as r_3^i, respectively, if the literal of x_i is negated).

Finally, it only remains to colour the clause section. Consider, independently of others, a clause $c_m = (\ell_{m_1} \vee \ell_{m_2} \vee \ell_{m_3})$. By the assumption of a not-all-equal satisfying assignment of Φ, the points $p_5^{m_1}, p_5^{m_2}, p_5^{m_3}$ are not all red and not all blue. Up to symmetry, $p_5^{m_3}$ is red, and so we colour s_3^m blue and s_4^m yellow. Then one of $p_5^{m_1}, p_5^{m_2}$ is not red, and so we may use remaining red and orange to colour s_1^m, s_2^m in a suitable order. Using the same rules for all clause gadgets, we do not get any "global" conflict since s_4^m would always be yellow and hence different from s_4^{m+1}, etc. Lastly, if the rightmost end (yellow) of the clause section conflicts with the leftmost end r_1^1 of the variable section, then we exchange yellow with orange in the whole clause section. This is a proper colouring of G, proving Claim 9.

The last step is to adjust the proof for $k > 6$. This is straightforward, and so we only sketch the small change: We expand the colour-fix gadget with additional $k - 6$ vertices

$a_7, \ldots, a_k$, to be coloured by more shades of the green colour. We analogously add $k-6$ new vertices to each copy gadget between p_1^j and p_2^j. All the arguments then remain the same. ◀

4 Conclusions

In this paper we have showed that the problem of deciding 6-colourability for visibility graphs of simple polygons, is NP-hard. We have also showed that the 4-colouring problem can be solved for visibility graphs of simple polygons, in polynomial time. However, the 5-colouring problem still remains open. Also, we would like to point out to the reader that the 4-colouring and 5-colouring problems on polygons with holes require further study.

References

1 Oswin Aichholzer, Greg Aloupis, Erik D. Demaine, Martin L. Demaine, Vida Dujmovic, Ferran Hurtado, and Günter Rote, André Schulz, Diane L. Souvaine, and Andrew Winslow Anna Lubiw. Convexifying polygons without losing visibilities. In *CCCG*, 2011.

2 Martin Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984. doi:10.1016/0166-218X(84)90073-8.

3 Matthew Babbitt, Jesse Geneson, and Tanya Khovanova. On k-visibility graphs. *Journal of Graph Algorithms and Applications*, 19(1):345–360, 2015.

4 Andreas Bärtschi, Subir Kumar Ghosh, Matúš Mihalák, Thomas Tschager, and Peter Widmayer. Improved bounds for the conflict-free chromatic art gallery problem. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 144:144–144:153, New York, NY, USA, 2014.

5 A. A. Diwan and B. Roy. On Colouring Point Visibility Graphs. *Proceedings of the 3rd Conference on Algorithms and Discrete Applied Mathematics*, pages 156–165, 2017 (Manuscript updated later - *arXiv:1610.00952*.

6 Sándor P. Fekete, Stephan Friedrichs, Michael Hemmer, Joseph B. M. Mitchell, and Christiane Schmidt. On the chromatic art gallery problem. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada, 2014*, 2014.

7 Subir Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, New York, NY, USA, 2007.

8 J. Hershberger. Finding the visibility graph of a polygon in time proportional to its size. *Algorithmica*, 4:141–155, 1989.

9 Frank Hoffmann, Klaus Kriegel, Subhash Suri, Kevin Verbeek, and Max Willert. Tight Bounds for Conflict-Free Chromatic Guarding of Orthogonal Art Galleries. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 421–435, 2015.

10 J. Kára, A. Pór, and D. R. Wood. On the Chromatic Number of the Visibility Graph of a Set of Points in the Plane. *Discrete and Computational Geometry*, 34(3):497–506, 2005.

11 Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

12 D. T. Lee and A. K. Lin. Computational Complexity of Art Gallery Problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

13 Yaw-Ling Lin and Steven Skiena. Complexity aspects of visibility graphs. *International Journal of Computational Geometry and Applications*, 5:289–312, 1995.

14 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

15 J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987.

16 J. O'Rourke and K. Supowit. Some NP-hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29:181–190, 1983.

17 Joseph O'Rourke and Ileana Streinu. The vertex-edge visibility graph of a polygon. *Computational Geometry*, 10(2):105–120, 1998.

18 F. Pfender. Visibility Graphs of Point Sets in the Plane. *Discrete and Computational Geometry*, 39(1):455–459, 2008.

19 S. K. Ghosh and T. Shermer and B. K. Bhattacharya and P. P. Goswami. Computing the maximum clique in the visibility graph of a simple polygon. *Journal of Discrete Algorithms*, 5:524–532, 2007.

20 T. Shermer. Hiding people in polygons. *Computing*, 42:109–131, 1989.

Vertex Deletion Problems on Chordal Graphs*†

Yixin Cao[1], Yuping Ke[2], Yota Otachi[3], and Jie You[4]

1 **Department of Computing, Hong Kong Polytechnic University, Hong Kong, China**
yixin.cao@polyu.edu.hk
2 **Department of Computing, Hong Kong Polytechnic University, Hong Kong, China**
yu.ke@polyu.edu.hk
3 **Faculty of Advanced Science and Technology, Kumamoto University, Kumamoto, Japan**
otachi@cs.kumamoto-u.ac.jp
4 **School of Information Science and Engineering, Central South University and Department of Computing, Hong Kong Polytechnic University, Hong Kong, China**
jie.you@polyu.edu.hk

Abstract

Containing many classic optimization problems, the family of vertex deletion problems has an important position in algorithm and complexity study. The celebrated result of Lewis and Yannakakis gives a complete dichotomy of their complexity. It however has nothing to say about the case when the input graph is also special. This paper initiates a systematic study of vertex deletion problems from one subclass of chordal graphs to another. We give polynomial-time algorithms or proofs of NP-completeness for most of the problems. In particular, we show that the vertex deletion problem from chordal graphs to interval graphs is NP-complete.

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity, G.2.2 Graph Theory

Keywords and phrases vertex deletion problem, maximum subgraph, chordal graph, (unit) interval graph, split graph, hereditary property, NP-complete, polynomial-time algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.22

1 Introduction

Generally speaking, a vertex deletion problem asks to transform an input graph to a graph in a certain class by deleting a minimum number of vertices. Many classic optimization problems belong to the family of vertex deletion problems, and their algorithms and complexity have been intensively studied. For example, the clique problem and the independent set problem are nothing but the vertex deletion problems to complete graphs and to edgeless graphs respectively. Most interesting graph properties are *hereditary*: If a graph satisfies this property, then so does every induced subgraph of it. For all the vertex deletion problems to hereditary graph classes, Lewis and Yannakakis [26] have settled their complexity once and for all with a dichotomy result: They are either NP-hard or trivial. Thereafter algorithmic efforts

* A full version of the paper is available at https://arxiv.org/abs/1707.08690.

† Supported in part by the Hong Kong Research Grants Council (RGC) under grants 252026/15E and 152261/16E, and the National Natural Science Foundation of China (NSFC) under grant 61572414.

were mostly focused on the nontrivial ones, and the major approaches include approximation algorithms [27], parameterized algorithms [6], and exact algorithms [15].

Chordal graphs make one of the most important graph classes. Together with many of its subclasses, it has played important roles in the development of structural graph theory. (We defer their definitions to the next section.) Many algorithms have been developed for vertex deletion problems to chordal graphs and its subclasses,—most notably (unit) interval graphs, cluster graphs, and split graphs; see, e.g., [17, 4, 10, 9, 8, 33, 12, 25, 1] for a partial list. After the long progress of algorithmic achievements, some natural questions arise: What is the complexity of transforming a chordal graph to a (unit) interval graph, a cluster graph, a split graph, or a member of some other subclass of chordal graphs? It is quite surprising that this type of problems has not been systematically studied, save few concrete results, e.g., the polynomial-time algorithms for the clique problem, the independent set problem, and the feedback vertex set problem (the object class being forests) [21, 32].

The same question can be asked for other pair of source and object graph classes. The most important source classes include planar graphs [20, 18, 16], bipartite graphs [31], and degree-bounded graphs [19]. As one may expect, with special properties imposed on input graphs, the problems become easier, and some of them may not remain NP-hard. Unfortunately, a clear-cut answer to them seems very unlikely, since their complexity would depend upon both the source class and the object class. Indeed, some are trivial (e.g., vertex cover on split graphs), some remain NP-hard (e.g., vertex cover on planar graphs), while some others are in P but can only be solved by very nontrivial polynomial-time algorithms (e.g., vertex cover on bipartite graphs).

Throughout the paper we write the names of graph classes in small capitals; e.g., CHORDAL and BIPARTITE stand for the class of chordal graphs and the class of bipartite graphs respectively. We use $\mathcal{C}$, commonly with subscripts, to denote an unspecified hereditary graph class, and use $\mathcal{C}_1 \to \mathcal{C}_2$ to denote the vertex deletion problem from class $\mathcal{C}_1$ to class $\mathcal{C}_2$:

Given a graph G in $\mathcal{C}_1$, find a minimum set $V_- \subseteq V(G)$ such that $G - V_-$ is in $\mathcal{C}_2$.

It is worth noting that $\mathcal{C}_2$ may or may not be a subclass of $\mathcal{C}_1$, and when it is not, the problem is equivalent to $\mathcal{C}_1 \to \mathcal{C}_1 \cap \mathcal{C}_2$: Since $\mathcal{C}_1$ is hereditary, $G - V_-$ is necessarily in $\mathcal{C}_1$. For almost all classes $\mathcal{C}$, the complexity of problems PLANAR $\to \mathcal{C}$ and BIPARTITE $\to \mathcal{C}$ has been answered in a systematical manner [26, 31], while for most graph classes $\mathcal{C}$, the complexity of problem DEGREE-BOUNDED $\to \mathcal{C}$ has been satisfactorily determined [19].

Apart from CHORDAL, we would also consider vertex deletion problems on its subclasses. Therefore, our purpose in this paper is a focused study on the algorithms and complexity of $\mathcal{C}_1 \to \mathcal{C}_2$ with both $\mathcal{C}_1$ and $\mathcal{C}_2$ being subclasses of CHORDAL. Since it is generally acknowledged that the study of chordal graphs motivated the theory of perfect graphs [24, 2], the importance of chordal graphs merits such a study from the aspect of structural graph theory. However, our main motivation is from the recent algorithmic progress in vertex deletion problems. It has come to our attention that to transform a graph to class $\mathcal{C}_1$, it is frequently convenient to first make it a member of another class $\mathcal{C}_2$ that contains $\mathcal{C}_1$ as a proper subclass, followed by an algorithm for the $\mathcal{C}_2 \to \mathcal{C}_1$ problem [29, 9, 7, 33].

There being many subclasses of CHORDAL, the number of problems fitting in our scope is quite prohibitive. The following simple observations would save us a lot of efforts.

▶ **Proposition 1.** *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two graph classes.*
(1.) *If the $\mathcal{C}_1 \to \mathcal{C}_2$ problem is in P, then so is $\mathcal{C} \to \mathcal{C}_2$ for any subclass $\mathcal{C}$ of $\mathcal{C}_1$.*
(2.) *If the $\mathcal{C}_1 \to \mathcal{C}_2$ problem is NP-complete, then so is $\mathcal{C} \to \mathcal{C}_2$ for any superclass $\mathcal{C}$ of $\mathcal{C}_1$.*

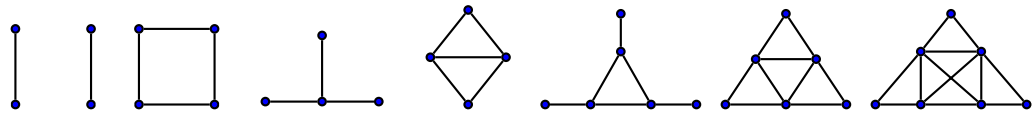


Figure 1 Small subgraphs: $2K_2$, C_4, claw ($K_{1,3}$), diamond, net, tent, rising sun.

For example, the majority of our hardness results for problems CHORDAL $\to \mathcal{C}$ are obtained by proving the hardness of SPLIT $\to \mathcal{C}$. Indeed, this is very natural as in literature, most (NP-)hardness of problems on chordal graphs is proved on split graphs, e.g., dominating set [3], Hamiltonian path [28], and maximum cut [5]. The most famous exception is probably the pathwidth problem, which can be solved in polynomial time on split graphs but becomes NP-complete on chordal graphs [23]. No problem like this surfaces during our study, though we do have the following hardness result proved directly on chordal graphs, for which we have no conclusion on split graphs.

▶ **Theorem 2.** *Let F be a biconnected chordal graph. If F is not complete, then the* CHORDAL $\to$ F-FREE *problem is NP-complete.*

Another simple observation of common use to us is about complement graph classes. The *complement* $\overline{G}$ of graph G is defined on the same vertex set $V(G)$, where a pair of distinct vertices u and v is adjacent in $\overline{G}$ if $uv \notin E(G)$. It is easy to see that the complement of $\overline{G}$ is G. In Figure 1, for example, the net and the tent are the complements of each other. The *complement* of a graph class $\mathcal{C}$, denoted by $\overline{\mathcal{C}}$, comprises all graphs whose complements are in $\mathcal{C}$; e.g., the complement of COMPLETE SPLIT is $\{2K_2, P_3\}$-FREE. A graph class $\mathcal{C}$ is *self-complementary* if it is its own complement, i.e., a graph $G \in \mathcal{C}$ if and only if $\overline{G} \in \mathcal{C}$. For example, both SPLIT and THRESHOLD are self-complementary. As usual, n denotes the number of vertices in the input graph. Note that we need an n^2 item because it takes $O(n^2)$ time to compute the complement of a graph.

▶ **Proposition 3.** *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two graph classes. If the $\mathcal{C}_1 \to \mathcal{C}_2$ problem can be solved in $f(n)$ time, then the $\overline{\mathcal{C}}_1 \to \overline{\mathcal{C}}_2$ problem can be solved in $O(f(n) + n^2)$ time.*

Our results (besides Theorem 2) are summarize in Figure 2. Unfortunately, we have to leave the complexity of some problems open, particularly CHORDAL $\to$ CLUSTER, CHORDAL $\to$ UNIT INTERVAL, and INTERVAL $\to$ UNIT INTERVAL.

Let us also mention the approximation algorithms. All the problems have constant-ratio approximations, which follow from either [7, 8] or the general observation of Lund and Yannakakis [27]. On the other hand, none of the NP-complete problems admits a polynomial-time approximation scheme.

2 Preliminaries

All graphs discussed in this paper are undirected and simple. A graph G is given by its vertex set $V(G)$ and edge set $E(G)$, whose cardinalities will be denoted by n and m respectively.

For a subset $X \subseteq V(G)$, denote by $G[X]$ the subgraph induced by X, and by $G - X$ the subgraph $G[V(G) \setminus X]$; we use $E(X)$ as a shorthand for $E(G[X])$, i.e., all edges among vertices in X. For a subset $E_- \subseteq E(G)$ of edges, we use $G - E_-$ to denote the subgraph with vertex set $V(G)$ and edge set $E(G) \setminus E_-$. We write $G - v$ and $G - e$ instead of $G - \{v\}$ and $G - \{e\}$ for $v \in V(G)$ and $e \in E(G)$ respectively.

For $\ell \geq 2$, we use P_ℓ, K_ℓ, and I_ℓ to denote an induced path, a clique, and an independent set, respectively, on ℓ vertices. For $\ell \geq 4$, we use C_ℓ to denote an induced cycle on ℓ vertices;

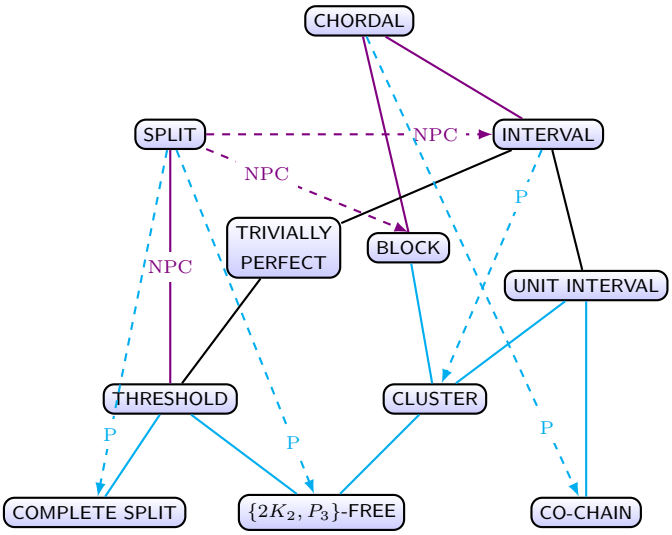


Figure 2 A summary of major graph classes studied by this paper and our results. Two classes are connected by a solid edge when the lower one is a subclass of the higher one. A directed dashed edge from $\mathcal{C}_1$ to $\mathcal{C}_2$ is used when $\mathcal{C}_2$ is not an immediate subclass of $\mathcal{C}_1$. We omit here results implied by Proposition 1; e.g., $\mathcal{C} \to$ CO-CHAIN is in P for all $\mathcal{C}$, and CHORDAL $\to \mathcal{C}$ is NP-complete when $\mathcal{C}$ is THRESHOLD, BLOCK, or INTERVAL. The cyan, violet, and black edges indicate that the complexity of the representing problems is in P, NP-complete, and unknown, respectively.

such a cycle is also called a *hole*. Some small graphs that will be used in this paper are depicted in Figure 1. Note that C_4 and $2K_2$ are complements to each other, while the complements of P_4 and C_5 are themselves.

We say that a graph G *contains* a subgraph F if F is isomorphic to some induced subgraph of G. A graph is *F-free* if it does not contain F; for a set $\mathcal{F}$ of graphs, a graph G is *$\mathcal{F}$-free* if it is F-free for every $F \in \mathcal{F}$. Each set $\mathcal{F}$ defines a hereditary graph class, and every hereditary graph class can be defined as such; in other words, for any hereditary graph class $\mathcal{C}$, there is a (possibly infinite) set $\mathcal{F}$ of subgraphs such that a graph $G \in \mathcal{C}$ if and only if it is $\mathcal{F}$-free. Each graph F in $\mathcal{F}$ is usually assumed to be minimal, in the sense that F is not in $\mathcal{C}$ but every proper induced subgraph of F is; they are called the *minimal obstructions* of $\mathcal{C}$. One should note that a minimal obstruction of a graph class may not be a minimal obstruction of its subclass; e.g., the minimal obstruction C_5 of SPLIT is not a minimal obstruction of THRESHOLD, because C_5 contains the non-threshold graph P_4 as a proper induced subgraph.

The vertex deletion problem with object class $\mathcal{C}$ can also be defined as finding a maximum subgraph in the class $\mathcal{C}$. For example, both vertex cover and independent set refer to the vertex deletion problem to the class EDGELESS, which is exactly the K_2-free graphs. Although these formulations may behave different with respect to approximation, they are the same for our purpose. We may use both formulations interchangeably, dependent on which is more convenient in the context. Yet another way to view the vertex deletion problem toward property $\mathcal{F}$-free is to find a minimum set of vertices from a graph to hit all its induced subgraphs in $\mathcal{F}$.

We now define the graph classes we are going to study. Although the containment relationships of all the graph classes to be studied can be readily checked with their obstruction characterizations, sometimes it would be far more informative and inspiring if we look at them from the lens of the definitions and/or geometric representations of these graph classes.

A graph is *chordal* if every cycle of length larger than three has a chord, i.e., an edge between two non-consecutive vertices of the cycle. A graph is an *interval graph* if its vertices can be assigned to intervals on the real line such that there is an edge between two vertices if and only if their corresponding intervals intersect, and a *unit interval graph* if all the intervals have the same length. A graph G is a *trivially perfect graph* if for every induced subgraph of G, the size of the largest independent set is equivalent to the number of all maximal cliques [22]. Chordal graphs are precisely graphs that are intersection graphs of subtrees of a tree, while interval graphs are intersection graphs of sub-paths of a path. Therefore, INTERVAL $\subset$ CHORDAL. A trivially perfect graph can be represented by a set of *non-overlapping* intervals; in other words, if two intervals intersect, then one is contained in the other. Therefore, TRIVIALLY PERFECT $\subset$ INTERVAL.

A graph is a *cluster graph* if every component is a clique. A graph is a *block graph* if the deletion of all cut vertices leaves a cluster graph. It is known that a graph is $\{2K_2, P_3\}$-free if it is a cluster graph of which at most one clique is nontrivial, i.e., having more than one vertex. It is immediate from their definitions that $\{2K_2, P_3\}$-FREE $\subset$ CLUSTER $\subset$ BLOCK. Moreover, block graphs are precisely those chordal graph of which any two maximal cliques share at most one vertex.

A graph is a *split graph* if its vertices can be partitioned into a clique C and an independent set I, and a *complete split graph* if every vertex in C is adjacent to all vertices in I; we use $C \uplus I$ to denote the split partition. Note that either of the two sets may be empty. A graph G is a *threshold graph* if there is a real number t, the so-called *threshold*, and an assignment $f : V(G) \to \mathbb{R}$ such that $uv \in E(G)$ if and only if $f(u) + f(v) \geq t$ [11]. It is easy to verify that COMPLETE SPLIT $\subset$ THRESHOLD $\subset$ SPLIT: The first can be witnessed by $t = 1$ and assignment $f(v) = 1$ if $v \in C$ and 0 otherwise; and the second by the clique partition $\{v : f(v) \geq t/2\} \uplus \{v : f(v) < t/2\}$. Further, if we order the vertices in the independent set I of a threshold graph such that $f(v_1) \leq \cdots \leq f(v_{|I|}) < t/2$, then $N(v_1) \subseteq \cdots \subseteq N(v_{|I|})$. Likewise, there is an ordering of vertices $u_1, \ldots, u_{|C|}$ in C such that $N[u_1] \subseteq \cdots \subseteq N[u_{|C|}]$.

The reader may have noticed the striking resemblance between split graphs and bipartite graphs. Indeed, if we add edges to make one side of a bipartite graph into a clique, we end with a split graph; or equivalently, given a split graph G with split partition $C \uplus I$, the subgraph $G - E(C)$ is bipartite. Clearly, $G - E(C)$ is a complete bipartite graph if and only if G is a complete split graph. If G is a threshold graph, then $G - E(C)$ is a *chain graph* [31, 30]. Finally, CO-CHAIN denotes the complement of CHAIN.

Recall that Yannakakis [31] has given a dichotomy on the vertex deletion problem from bipartite graphs. Inspired by this and the aforementioned connection between bipartite graphs and split graphs, a natural attempt at problems SPLIT $\to \mathcal{C}$ would be reducing them to the corresponding problem on bipartite graphs (for algorithms) or the other way (for hardness results). This approach however turns out to be less straightforward as one may expect. See the full version for discussions.

3 Algorithmic results

This section gives the polynomial-time algorithms. Our focus would be laid on the use of structural properties, and if possible, we would present the simplest algorithms without elaborating on the implementation details. These problems may have more efficient algorithms, and with more complex data structures and algorithmic finesses, some of them may even be solved in linear time.

Our first two results are on split graphs, for which we need to put split partitions under

0. $\mathcal{S} \leftarrow \emptyset$;
1. build a bipartite graph G' by removing all edges among C from G;
2. find a minimum vertex cover of G', and add it to $\mathcal{S}$;
3. **for each** $v \in I$ **do**
   find a minimum vertex cover X of $G' - (C \setminus N(v)) - v$;
   add $X \cup (C \setminus N(v))$ to $\mathcal{S}$;
4. **return** a set in $\mathcal{S}$ with the minimum cardinality.

Figure 3 Algorithm for SPLIT $\rightarrow \{2K_2, P_3\}$-FREE.

scrutiny. Let $C \uplus I$ be a split partition of a split graph G. If some vertex in I is completely adjacent to C, then we can move such a vertex v to C to make another split partition $C' = C \cup \{v\}$ and $I' = I \setminus \{v\}$. Note that the vertex v may not be unique, and the resulting graphs by moving them would be isomorphic. Moreover, after such a move, no vertex of I' can be completely adjacent to C'. The following proposition fully characterizes split graphs with more than one different split partition.

▶ **Proposition 4.** *Let G be a split graph with at least two split partitions, and let $C \uplus I$ and $C' \uplus I'$ be two different split partitions of G.*

(i) *The difference between $|C|$ and $|C'|$ is at most 1.*

(ii) *If $|C| = |C'| + 1$, then C is a maximum clique, and I' is a maximum independent set of G; moreover, $C' \subset C$.*

(iii) *If $|C| = |C'|$, then $G - E(C)$ and $G - E(C')$ are isomorphic.*

As a result, a split graph has either one or two essentially distinct split partitions. On the other hand, of all split partitions of a complete bipartite graph, only one, whose independent set is the largest, satisfies the definition of complete bipartite graphs, and we will exclusively refer to it when we are discussing a complete split graph.

Let G be a split graph with split partition $C \uplus I$ and let $\underline{G}$ be a $\{2K_2, P_3\}$-free subgraph of G. If $\underline{G}$ has edges, all of them must be in the same nontrivial clique. At most one vertex of this clique can be from I; therefore, all other vertices of I either are deleted or become isolated in $\underline{G}$. In other words, for each other vertex v in I, either v or all its neighbors have to be deleted.

▶ **Theorem 5.** *The* SPLIT $\rightarrow \{2K_2, P_3\}$-FREE *problem is in P.*

Proof. Let G be the input graph to the SPLIT $\rightarrow \{2K_2, P_3\}$-FREE problem and let $C \uplus I$ be a split partition of G. We use the algorithm in Figure 3 to find a minimum solution to G. To argue its correctness, we show that (i) every set in $\mathcal{S}$, added in step 2 or 3, is a solution to G, and (ii) at least one of them is minimum. For (i), it is easy to verify that any vertex cover of $G' = G - E(C)$ is a solution: There is no edge between C and I after its deletion. The situation in step 3 is similar; note that $N[v] \uplus (I \setminus \{v\})$ is a split partition of $G - \big(C \setminus N(v)\big)$.

Let V_- be a minimum solution to G. In the first case, every vertex $v \in I \setminus V_-$ is isolated in $G - V_-$. In other words, V_- contains a vertex cover of $G' = G - E(C)$, and then the solution found by step 2 is already the minimum. Henceforth we assume that there exists a vertex $v \in I \setminus V_-$ such that $N(v) \not\subseteq V_-$. Since any vertex $u \in N(v)$ and $w \in C \setminus N(v)$ induce a P_3 with v, in this case all vertices in $C \setminus N(v)$ must be in V_-. Note that the vertex v is unique: If two vertices in $I \setminus V_-$ have neighbors in $C \setminus V_-$, then they are in a non-clique component. Therefore, after removing $C \setminus N(v)$ and v from the graph, it reduces to the first case. This justifies step 3.

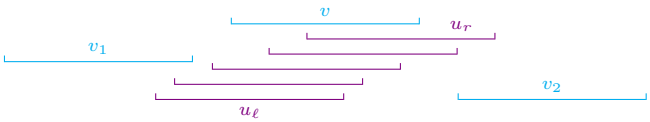


Figure 4 A connected split graph with split partition $C \uplus I$ that is also a unit interval graph. Violet intervals are for vertices in C and cyan for I. Note that the vertex v from I is completely adjacent to C.

The algorithm makes $O(n)$ calls to an algorithm for the bipartite vertex cover problem, each taking $O(m\sqrt{n})$ time, and hence the whole algorithm runs in $O(mn\sqrt{n})$ time. ◀

Noting that SPLIT ∩ CLUSTER is precisely $\{2K_2, P_3\}$-FREE, we can apply the algorithm of Theorem 5 to the SPLIT → CLUSTER problem. Moreover, since SPLIT is self-complementary, while the complement of $\{2K_2, P_3\}$-FREE is COMPLETE SPLIT, it follows from Proposition 3 that the SPLIT → COMPLETE SPLIT problem is also in P.

▶ **Corollary 6.** *Problems* SPLIT → CLUSTER *and* SPLIT → COMPLETE SPLIT *are in* P.

A similar observation can be used to solve the SPLIT → UNIT INTERVAL problem. We start from a simple property of connected graphs in SPLIT ∩ UNIT INTERVAL.

▶ **Proposition 7.** *Let G be a connected split graph and let $C \uplus I$ be a split partition of G. If G is a unit interval graph, then $|I| \leq 3$, and the equality holds only when there is a vertex $v \in I$ adjacent to all vertices in C.*

Proof. We prove $|I| \leq 2$ if C is a maximum clique of G, and then the proposition follows from Proposition 4(i). Let u_ℓ and u_r be the vertices in C with respectively the leftmost and rightmost intervals. Suppose for contradiction $|I| > 2$. Let v_1 and v_2 be the vertices in I with respectively the leftmost and rightmost intervals. Then $\mathtt{lp}(u_\ell) < \mathtt{rp}(v_1) < \mathtt{lp}(u_r) < \mathtt{rp}(u_\ell) < \mathtt{lp}(v_2) < \mathtt{rp}(u_r)$, where the second and the fourth inequalities follow from that C is a maximum clique, and the others from the selections of the four vertices. Since G is connected, the interval for any other vertex v in $I \setminus \{v_1, v_2\}$, which is nonempty, has to lie in $(\mathtt{rp}(v_1), \mathtt{lp}(v_2))$. But then it has to contain $[\mathtt{lp}(u_r), \mathtt{rp}(u_\ell)]$, and $\{v\} \cup C$ is a clique, contradicting that C is a maximum clique of G. ◀

Similar as Theorem 5, our algorithm for SPLIT → UNIT INTERVAL separates into two cases, based on whether there is a vertex of $I \setminus V_-$ adjacent to all vertices in $C \setminus V_-$.

▶ **Theorem 8.** *The* SPLIT → UNIT INTERVAL *problem is in* P.

Proof. Let G be the input graph to the SPLIT → UNIT INTERVAL problem and let $C \uplus I$ be a split partition of G. We use the algorithm in Figure 5 to find a solution. To argue its correctness, we show that all sets put into $\mathcal{S}$ in steps 1–4 are solutions to G, and at least one of them is minimum. It is clear for step 1. After the deletion of a solution found in step 2, only v in I remains adjacent to the remaining vertices of C. In step 3, only v_1 and v_2 from I can remain adjacent to vertices in C. In step 3.2, no vertex in C is adjacent to both v_1 and v_2; in step 3.3, every vertex in C is adjacent to at least one of v_1 and v_2. In either case, it is easy to verify that the graph is a unit interval graph by building a unit interval model directly. Step 4 follows from the same argument as above: After the deletion of $C \setminus N(v)$, it reduces to one of the three previous steps.

0. $\mathcal{S} \leftarrow \emptyset$;
1. solve the SPLIT $\rightarrow$ $\{2K_2, P_3\}$-FREE problem on G; add the solution to $\mathcal{S}$;
   $\backslash\backslash$ **case 1:**
2. **for each** $v \in I$ **do**
   find a minimum vertex cover of $G - v - E(C)$, and add it to $\mathcal{S}$;
3. **for each** $v_1, v_2 \in I$ **do**
3.1. $G' \leftarrow G - \{v_1, v_2\} - E(C)$;
3.2. find a minimum vertex cover of $G' - N(v_1) \cap N(v_2)$,
   and add its union with $N(v_1) \cap N(v_2)$ to $\mathcal{S}$;
3.3. find a minimum vertex cover of $G' - C \setminus \big(N(v_1) \cup N(v_2)\big)$,
   and add its union with $C \setminus \big(N(v_1) \cup N(v_2)\big)$ to $\mathcal{S}$;
   $\backslash\backslash$ **case 2:**
4. **for each** $v \in I$ **do**
   $G'' \leftarrow G - \big(C \setminus N(v)\big)$ with split partition $N[v]$ and $I \setminus \{v\}$;
   solve G'' as case 1, but append $C \setminus N(v)$ to each solution found;
5. **return** a set in $\mathcal{S}$ with the minimum cardinality.

Figure 5 Algorithm for SPLIT $\rightarrow$ UNIT INTERVAL.

Let V_- be a minimum solution to G. If $G - V_-$ is $\{2K_2, P_3\}$-free, then the solution found by step 1 is the minimum. Henceforth we assume that $G - V_-$ contains a non-clique component U; note that such a component contains all vertices in $C \setminus V_-$ and hence is unique.

In the first case, every vertex $v \in U \cap I$ has at least one non-neighbor in $C \setminus V_-$, i.e., $N(v) \setminus V_- \subset C \setminus V_-$. According to Proposition 7, $|U \cap I| \leq 2$. If $U \cap I = \{v\}$, then $G - (V_- \cup \{v\})$ is $\{2K_2, P_3\}$-free and the only nontrivial clique $U \setminus \{v\}$ is a subset of C; hence step 2 always find a minimum solution. In the rest of this case, $U \cap I$ has two different vertices; let them be v_1 and v_2. Since any $u_1 \in N(v_1) \cap N(v_2)$ and $u_2 \in C \setminus \big(N(v_1) \cup N(v_2)\big)$ induce a claw with $\{v_1, v_2\}$, at least one of the two sets needs to be empty or completely contained in V_-. Steps 3.2 and 3.3 take care of these two situations separately.

We are now in the second case, where $C \setminus V_- \subseteq N(v)$ for some vertex $v \in I \setminus V_-$; in other words, V_- contains all vertices in $C \setminus N(v)$. There might be two of such vertices, when we can take v to be either of them. Clearly, $N[v]$ and $I \setminus \{v\}$ is then a split partition of $G'' = G - \big(C \setminus N(v)\big)$, which has a solution $V_- \setminus \big(C \setminus N(v)\big)$. Moreover, under this new split partition, we reduce it to the first case.

The algorithm makes $O(n^3)$ calls to the algorithm for the bipartite vertex cover problem, each taking $O(m\sqrt{n})$ time, and hence the whole algorithm runs in $O(mn^{3.5})$ time. ◀

We now turn to problems whose inputs are interval graphs, for which we rely on interval models. Recall that an interval model for an interval graph is a set of intervals representing its vertices. In this paper, all intervals are closed. An interval model can be specified by the $2n$ endpoints for the n intervals, the interval for vertex v being by $[\mathtt{lp}(v), \mathtt{rp}(v)]$.

For the UNIT INTERVAL $\rightarrow$ COMPLETE SPLIT problem, the clique is from some maximal clique of the input graph G and can be enumerated. On the other hand, according to Proposition 7, there are at most three vertices in the independent set, which can be easily found. However, for interval graphs, it can be more complicated.

▶ **Theorem 9.** *Problems* INTERVAL $\rightarrow$ COMPLETE SPLIT *and* INTERVAL $\rightarrow$ CLUSTER *are in P.*

Proof. We solve both problems by finding the maximum subgraphs, for which we work on interval models. Let us fix an interval model for the input graph G; we may assume without loss of generality that no distinct intervals can share an endpoint.

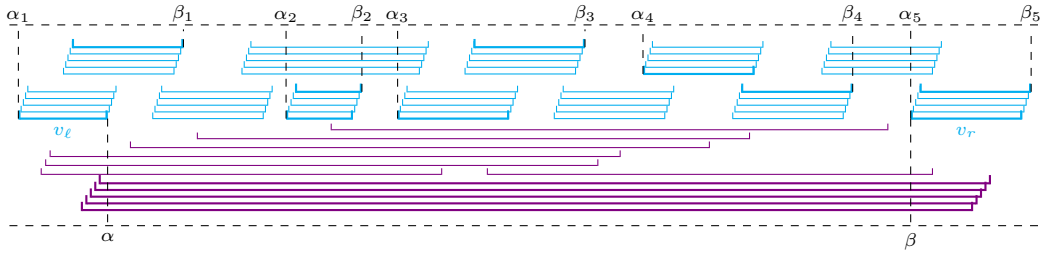


Figure 6 Given is an interval model for an interval graph G.
The top line is for the INTERVAL → CLUSTER problem. A maximum cluster subgraph of G has five cliques, each specified by a pair of α_i and β_i. (In this example, the maximum clique in each range $[\alpha_i, \beta_i]$ comprises all intervals in this range.)
The bottom line is for the INTERVAL → COMPLETE SPLIT problem. A maximum complete split subgraph of G contains 12 vertices. The clique contains the vertices represented by the lowest five intervals, and the independent set contains v_ℓ and v_r, together with a maximum independent set of all intervals completely lying in $[\alpha, \beta]$.

For the INTERVAL → COMPLETE SPLIT problem, we consider a maximum complete split subgraph $G[U]$. It is trivial if $G[U]$ is a clique; hence we assume otherwise. Let $C \uplus I$ be the split partition of $G[U]$, and let

$$\alpha = \mathtt{rp}(v_\ell) = \min_{v \in I} \mathtt{rp}(v) \text{ and } \beta = \mathtt{lp}(v_r) = \max_{v \in I} \mathtt{lp}(v).$$

Note that $|I| \geq 2$, as otherwise $G[U]$ is a clique; hence $v_\ell \neq v_r$ and $\alpha < \beta$. See Figure 6. It is easy to see that a vertex is in C if and only if its interval fully contains $[\alpha, \beta]$; on the other hand, the maximality of U requires us to take all such vertices. The independent set I would then consists of v_ℓ, v_r, and a maximum independent set of the subgraph induced by intervals satisfying $\alpha < \mathtt{lp}(v) < \mathtt{rp}(v) < \beta$. There are $O(n^2)$ pairs of indices to enumerate, and for each pair, both the clique and a maximum independent set can be found in $O(n)$ time. The whole algorithm runs in $O(n^3)$ time.

We now consider the INTERVAL → CLUSTER problem. Suppose that $G[U]$ is a maximum cluster subgraph of G and that it has k cliques. For the ith clique B_i, we can find two endpoints

$$\alpha_i = \min_{v \in B_i} \mathtt{lp}(v) \text{ and } \beta_i = \max_{v \in B_i} \mathtt{rp}(v).$$

Then all intervals for vertices in B_i are completely contained in the interval $[\alpha_i, \beta_i]$. The k intervals defined as such are pairwise disjoint: There cannot be edges between two cliques in $G[U]$. Therefore, B_i must be a maximum clique in the subgraphs induced by $\{v : \alpha_i \leq \mathtt{lp}(v) < \mathtt{rp}(v) \leq \beta_i\}$, which can be found easily. See Figure 6. The problem can thus be reduced to find the k pairs of endpoints α_i and β_i.

We build another weighted interval model as follows. For each $\mathtt{lp}(v_\ell)$ and each $\mathtt{rp}(v_r)$ with $\mathtt{lp}(v_\ell) < \mathtt{rp}(v_r)$, possibly $v_\ell = v_r$, we add an interval $[\mathtt{lp}(v_\ell), \mathtt{rp}(v_r)]$, whose weight is set to be the size of maximum cliques in the subgraphs induced by $\{v : \mathtt{lp}(v_\ell) \leq \mathtt{lp}(v) < \mathtt{rp}(v) \leq \mathtt{rp}(v_r)\}$. We then find a set of pairwise disjoint intervals with the maximum weight sum (or equivalently, a maximum-weight independent set of the weighted interval graph represented by the new interval model). All the steps can be done in polynomial time. ◀

It is easy to verify the following greedy algorithm solves the TREE → CLUSTER problem. We root the input graph at an arbitrary vertex, and work on any leaf at the lowest level: If it

has siblings (i.e., its parent has degree larger than 2), then delete its parent and put it into the solution; otherwise the parent of its parent. As we see below, a similar idea would enable us to solve the BLOCK → CLUSTER problem. Recall that a *block* (also known as biconnected component) of a graph G is a maximal biconnected subgraph of G. The *block-cut tree* of a block graph has a vertex for each block and for each cut vertex, and an edge for each pair of a block and a cut vertex that belongs to that block. Note that every block of a block graph is a clique.

▶ **Theorem 10.** *The* BLOCK → CLUSTER *problem can be solved in polynomial time.*

Proof. We construct the block-cut tree T of the input graph G. A cut vertex v of G is denoted by the same label in T, while for a block vertex u of T, we use $B(u)$ to denote the vertices in the block of G. We arbitrarily root T at some block vertex. Note that all leaves of T are block vertices, and their neighbors are not; this invariant will be maintained during our algorithm. Until the tree becomes empty, the algorithm always picks a leaf vertex u at the lowest level. Let v be its parent. If v has other children, we remove v and its children from T and put v in the solution V_-. In the rest u is the only child of v; let u' be the parent of v, and let v' be the parent of u'. If at least one vertex in the clique $B(u')$ is not a cut vertex, then we remove v, u from T and put v in V_-. Otherwise, we remove the subtree rooted at v' from T; we put $B(u') \setminus \{v\}$ into the solution, and for each other child u_i of v' that is not a leaf, we solve the subgraph induced by $B(u_i)$ and its children. The correctness is quite straightforward, so we omit here. ◀

The proofs of the following results are left in the full version.

▶ **Theorem 11.** *The* CHORDAL → CO-CHAIN *problem is in P.*

▶ **Theorem 12.** *For any $p > 1$, the* CHORDAL → K_p-FREE *problem is in P.*

▶ **Theorem 13** ([14])**.** *The* CHORDAL → SPLIT *problem is in P.*

4 Hardness

We now turn to hardness results. Here the problems should be understood to be their decision versions: The input includes, apart from a graph G from $\mathcal{C}_1$, a positive integer k, and the problem is to decide whether G can be made a graph in $\mathcal{C}_2$ by deleting at most k vertices. All of them are in NP because all the concerned graph classes can be recognized in polynomial time. Our first hardness result, on SPLIT → THRESHOLD, follows easily from the results of Yannakakis [31] on bipartite graphs. Recall that a bipartite graph is not a chain graph if and only if it contains some $2K_2$, and a split graph is not a threshold graph if and only if it contains some P_4.

▶ **Lemma 14.** *The* SPLIT → THRESHOLD *problem is NP-complete.*

Proof. Let G be a bipartite graph with partition C and I. We add all possible edges among C to make it a clique. Let G' be the resulting graph, which is clearly a split graph, witnessed by the split partition $C \uplus I$. We argue for every vertex set U that $G[U]$ is a chain graph, i.e., being $2K_2$-free, if and only if $G'[U]$ is a threshold graph, i.e., being P_4-free. Let X be any set of four vertices. If $G[X]$ is $2K_2$, then $|X \cap C| = |X \cap I| = 2$, but then $G'[X]$ would be a P_4. The other direction can be argued similarly. Since the BIPARTITE → CHAIN problem is NP-hard [31], the lemma follows. ◀

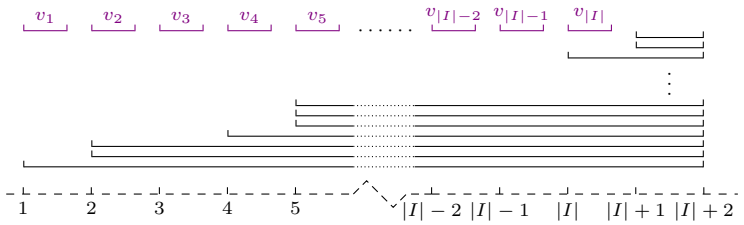


Figure 7 The interval model for a threshold graph given by (1).

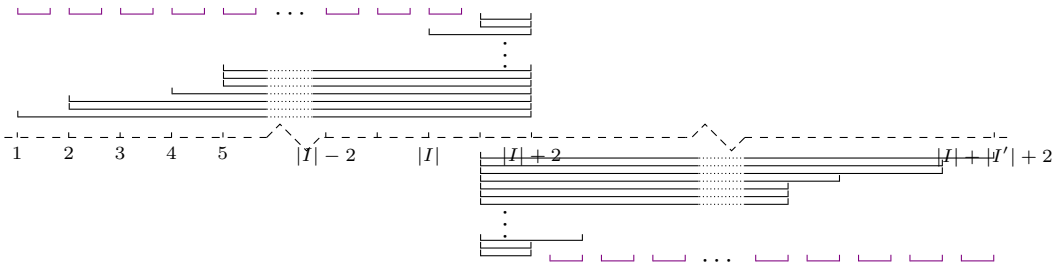


Figure 8 The interval model for $G_1 \bowtie G_2$.

Recall that every threshold graph is an interval graph, and this can be generalized as follows. Let G_1 and G_2 be two threshold graphs with split partitions $C \uplus I$ and $C' \uplus I'$ respectively. We let $G_1 \bowtie_{(C,C')} G_2$, or simply $G_1 \bowtie G'$ as in the rest of the paper the partitions are always clear from context, denote the graph obtained from them by adding all possible edges between C and C'—i.e., its vertex set and edge set are $V(G_1) \cup V(G_2)$ and $E(G_1) \cup E(G_2) \cup (C \times C')$ respectively. This is clearly a split graph with split partition $C \cup C'$ and $I \cup I'$. One can verify that $G_1 \bowtie G_2$ is also an interval graph by their obstructions as follows. A split graph that is not an interval graph has to contain a tent, a net, or a rising sun (see Figure 1). Each of them has three independent vertices, which have to be from $I \cup I'$, but a quick inspection of these three graphs will convince us that this cannot be possible.

▶ **Proposition 15.** *For any threshold graphs G_1, G_2, the graph $G_1 \bowtie G_2$ is an interval graph.*

A better way to look at Proposition 15 is probably through interval models. Let G be a threshold graph with split partition $C \uplus I$, and let vertices in I be ordered in a way that $N(v_1) \subseteq N(v_2) \subseteq \cdots \subseteq N(v_{|I|})$. We can build an interval model for G by setting intervals

$$\begin{aligned} [i, i+0.5] &\quad \text{for every } v_i \in I, \\ \big[\min\{i : v_i \in N(v)\}, |I|+2\big] &\quad \text{for every } v \in N(I), \text{ and} \\ \big[|I|+1, |I|+2\big] &\quad \text{otherwise } (i.e., v \in C \setminus N(I)). \end{aligned} \tag{1}$$

See Figure 7 for illustration.

An interval model for $G_1 \bowtie G_2$ can be built from the interval models for G_1 and G_2 by (i) keeping the intervals for G_1, and (ii) setting the interval to be $\big[|I|+|I'|+3-\mathtt{rp}(v), |I|+|I'|+3-\mathtt{lp}(v)\big]$ for each $v \in V(G_2)$. See Figure 8.

▶ **Theorem 16.** *The* SPLIT → INTERVAL *problem is NP-complete.*

Proof. It is clear that the problem is in NP. Let G be a split graph with split partition $C \uplus I$. We take a complete split graph G' with split partition $C' \uplus I'$, where $|C'| = |I'| = |C|$, and

let $H = G \bowtie G'$. We argue that (G, k) is a yes-instance of the SPLIT $\to$ THRESHOLD problem if and only if (H, k) is a yes-instance of the SPLIT $\to$ INTERVAL problem. Since both problems are trivial yes-instances when $k \geq |C|$, we may assume henceforth $k < |C|$.

Suppose that $G - V_-$, where $|V_-| \leq k$, is a threshold graph. According to Proposition 15, $(G - V_-) \bowtie G'$ is an interval graph. It is the same graph as $H - V_-$. Therefore, V_- is a solution of (H, k). This verifies the only if direction.

Now suppose that $H - V_-$, where $|V_-| \leq k$, is an interval graph. Suppose for contradiction that $\underline{G} = G - \big(V_- \cap V(G)\big)$ is not a threshold graph. Then $\underline{G}$ must contain some P_4; let it be $v_1u_1u_2v_2$. Since $\underline{G}$ is a split graph, we must have $u_1, u_2 \in C$ and $v_1, v_2 \in I$. On the other hand, by the assumption $k < |C|$, neither $C' \setminus V_-$ nor $I' \setminus V_-$ can be empty. Let $u \in C' \setminus V_-$ and $v \in I' \setminus V_-$. By the construction, the only edges between $\{u, v\}$ and $\{v_1, u_1, u_2, v_2\}$ are uu_1 and uu_2, but then these six vertices together induce a net in $H - V_-$, a contradiction. ◀

▶ **Corollary 17.** *The* CHORDAL $\to$ INTERVAL *problem is NP-complete.*

The last result is on the deletion of any biconnected subgraph from chordal graphs. Recall that a vertex v is *simplicial* in G if $N[v]$ is a clique. A graph is chordal if and only if we can make it empty by deleting simplicial vertices in the remaining graph [13].

▶ **Theorem 18.** *Let F be a biconnected chordal graph. If F is not complete, then the* CHORDAL $\to$ F-FREE *problem is NP-complete. Moreover, if F is a complete split graph with $|C| = 2$ and $|I| \geq 2$, then the* SPLIT $\to$ F-FREE *problem is NP-complete.*

Proof. We use the following reduction from the vertex cover problem. Let G be an input graph to the vertex cover problem, we conduct the following operations.

1. For each edge $uv \in E(G)$, add a distinct copy of F such that each of them uses uv as one of its edges. We say that u, v are the attachments for this copy of F.
2. Add all possible edges among $V(G)$ to make it complete.

Let G' be the obtained graph. To see that G' is chordal, we give an explicit way of eliminating simplicial vertices to make G' empty. A chordal graph either is a clique or contains two nonadjacent simplicial vertices; all vertices are simplicial when it is a clique. For each copy of F, we can find a simplicial vertex in $V(F) \setminus \{u, v\}$. We keep doing this, and then only vertices in $V(G)$ remain. They have been made a clique, and thus all of them simplicial.

We argue that G has a vertex cover of size k if and only if we can delete k vertices from G' to make it F-free. The following fact would be essential. We consider any copy X of F with attachments u and v. If we delete u or v, then the other becomes a cut vertex, and $X \setminus \{u, v\}$ are in different blocks from other vertices of $V(G')$. But any other copy of F, if it exists, must be completely contained in a block, and thus it cannot contains any vertex in X.

Suppose that V_- is a vertex cover of size k in G. We claim that $\underline{G} = G' - V_-$ has no copy of F. For each copy of F with attachments u and v. Therefore, a copy of F in $\underline{G}$, if one exists, has all its vertices from $V(G)$. But this is not possible because F is not a clique.

Suppose now that V_- is a solution to G' of size k. We may assume that V_- contains no new vertex: If it contains a vertex from a copy of F with attachments u and v, we can replace it by u. (Note that the new set remains a solution to G' because the aforementioned fact.) Since $G' - V_-$ does not contain F, each copy of it has at least one of the attachments in V_-. Therefore, each edge of G, at lest one end is in V_-, which means that V_- is a vertex cover of G. ◀

▶ **Corollary 19.** *Problems* SPLIT $\to$ BLOCK *and* CHORDAL $\to$ BLOCK *are NP-complete.*

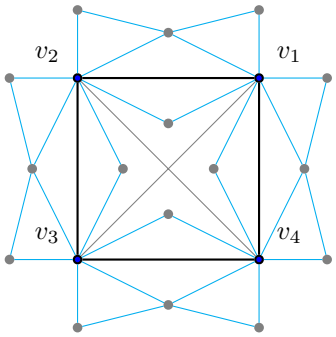


Figure 9 Reduction for Theorem 18, with F being a tent. The original graph G, drawn with blue vertices and thick edges, is a C_4. The new vertices are gray and new edges thin. The set $\{v_1, v_3\}$ is a solution to both problems.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback vertex set inspired kernel for chordal vertex deletion. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1383–1398. SIAM, 2017.

2 Claude Berge. *Some classes of perfect graphs.* Internat. Computation Centre, 1966.

3 Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Information processing letters*, 19(1):37–40, 1984.

4 Ivan Bliznets, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Largest chordal and interval subgraphs faster than 2^n. In *European Symposium on Algorithms*, pages 193–204. Springer, 2013.

5 Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 769–780. Springer, 1994.

6 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

7 Yixin Cao. Linear recognition of almost interval graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1096–1115. Society for Industrial and Applied Mathematics, 2016.

8 Yixin Cao. Unit interval editing is fixed-parameter tractable. *Information and Computation*, 253:109–126, 2017.

9 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(3):21, 2015.

10 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.

11 Václav Chvátal and Peter L. Hammer. Aggregation of inequalities in integer programming. *Annals of discrete mathematics*, 1:145–162, 1977.

12 Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: new fixed-parameter and exact exponential-time algorithms. *Information Processing Letters*, 113(5):179–182, 2013.

13 Gabriel Andrew Dirac. On rigid circuit graphs. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 25, pages 71–76. Springer, 1961.

14 Tinaz Ekim and Dominique de Werra. On split-coloring problems. *Journal of Combinatorial Optimization*, 10(3):211–225, 2005.

15 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 764–775. ACM, 2016.

16 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation and optimal FPT algorithms. *FOCS'12*, pages 470–479, 2012.

17 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.

18 Michael R. Garey and David S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

19 Michael R. Garey and David S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. *WH Free. Co., San Fr*, pages 90–91, 1979.

20 Michael R. Garey, David S. Johnson, and Larry Stockmeyer. Some simplified NP-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

21 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

22 Martin Charles Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24(1):105–107, 1978.

23 Jens Gusted. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

24 András Hajnal and János Surányi. Über die auflösung von graphen in vollständige teilgraphen. *Ann. Univ. Sci. Budapest, Eötvös Sect. Math*, 1:113–121, 1958.

25 Bart M.P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1399–1418. SIAM, 2017.

26 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

27 Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. *Automata, Languages and Programming*, pages 40–51, 1993.

28 Haiko Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1-3):291–298, 1996.

29 René Van Bevern, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Measuring indifference: Unit interval vertex deletion. In *WG*, pages 232–243. Springer, 2010.

30 Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

31 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981.

32 Mihalis Yannakakis and Fănică Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Information Processing Letters*, 24(2):133–137, 1987.

33 Jie You, Jianxin Wang, and Yixin Cao. Approximate association via dissociation. *Discrete Applied Mathematics*, 219:202–209, 2017.

A Lifting Theorem with Applications to Symmetric Functions*

Arkadev Chattopadhyay†[1] **and Nikhil S. Mande**‡[2]

1 **School of Technology and Computer Science, TIFR, Mumbai, India**
arkadev.c@tifr.res.in

2 **School of Technology and Computer Science, TIFR, Mumbai, India**
nikhil.mande@tifr.res.in

Abstract

We use a technique of "lifting" functions introduced by Krause and Pudlák [13], to amplify degree-hardness measures of a function to corresponding monomial-hardness properties of the lifted function. We then show that any symmetric function F projects onto a "lift" of another suitable symmetric function f. These two key results enable us to prove several results on the complexity of symmetric functions in various models, as given below:

1. We provide a characterization of the *approximate spectral norm* of symmetric functions in terms of the spectrum of the underlying predicate, affirming a conjecture of Ada et al. [1] which has several consequences[1] (cf. [1]).
2. We also characterize symmetric functions computable by quasi-polynomial sized Threshold of Parity circuits, resolving a conjecture of Zhang [24].
3. We show that the approximate spectral norm of a symmetric function f characterizes the (quantum and classical) bounded error communication complexity of $f \circ \mathsf{XOR}$.
4. Finally, we characterize the weakly-unbounded error communication complexity of symmetric XOR functions, resolving a weak form of a conjecture by Shi and Zhang [25].[2]

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Symmetric functions, lifting, circuit complexity, communication complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.23

1 Introduction

For any domain $\mathcal{A}$ and range $\mathcal{R}$, an n-variate function $f : \mathcal{A}^n \to \mathcal{R}$ is called *symmetric* if for all $x_1, \ldots, x_n \in \mathcal{A}$ and every permutation $\sigma \in S_n$, one has $f(x_1, \ldots, x_n) = f(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$. Symmetric functions are a very natural and basic class of functions, denoted by SYMM. There are several works about symmetric functions in different contexts in complexity theory that reveal their beautiful structure. As it is too numerous to list all of them out, we

* A full version of the paper is available at [6], https://arxiv.org/abs/1704.02537.
† Arkadev Chattopadhyay is partially supported by a Ramanujan fellowship of the DST.
‡ Nikhil S. Mande is partially supported by a TCS fellowship.
1 This has also been recently reported, after an early version of our manuscript was put up in the public domain, by Ada, Fawzi and Kulkarni [2] using a matrix theoretic result of Razborov [19], and other results.
2 The conjecture has been reported to be recently solved by independent works of Hatami and Qian [10] and Ada et al. [2]. Our techniques vary from theirs, a detailed comparison to related work can be found in Section 1.3.

very briefly recall a few here, some of which is relevant for this work: Paturi's famous theorem [17] characterizing the approximate degree of symmetric functions, Szegedy's [21] theorem characterizing functions that have bounded symmetric communication complexity making crucial use of symmetric functions, strong correlation bounds against low degree symmetric functions (polynomials) by Cai, Green and Thierauf [5], Razborov's theorem [19] characterizing the quantum bounded error communication complexity of $\mathsf{SYMM} \circ \mathsf{AND}$ and Sherstov's [20] theorem characterizing the unbounded error communication complexity of the same class of functions. More recently, and of particular relevance to this work, Shi and Zhang [25] characterized the (quantum) bounded-error complexity of $\mathsf{SYMM} \circ \mathsf{XOR}$ and Ada, Fawzi and Hatami [1] characterized the spectral norm of all symmetric functions. Shi and Zhang conjectured a certain characterization of the *unbounded-error* complexity of $\mathsf{SYMM} \circ \mathsf{XOR}$. Ada et al. conjectured a characterization of the *approximate* spectral norm of symmetric functions that in a way would extend Paturi's [17] characterization of the approximate degree of symmetric functions. Though these conjectures do not seem related on first glance, our work is motivated by them. In addition to proving the conjecture of Ada et al., we provide, among other things, the first characterization of the *weakly unbounded-error* communication complexity of $\mathsf{SYMM} \circ \mathsf{XOR}$. Both our results make use of a simple but somewhat surprising closure-like property of symmetric functions. The discovery of this property is one of our main technical contributions.

Krause and Pudlák [13] introduced a notion of 'lifting' functions to increase their hardness. Using this, they derived a technique to lower bound the sign monomial complexity (equivalently $\mathsf{THR} \circ \mathsf{XOR}$ circuit size) of a lifted function, f^{op}, in terms of the sign degree of f. As the lift of an AC^0 function can easily be seen to remain in AC^0, they were able to prove exponential lower bounds on the signed monomial complexity of a function in AC^0 using known sign degree lower bounds of AC^0 functions [15]. However, it is not clear how to use this lifting technique to prove lower bounds against other classes of functions. This lift is now more widely known as composition with the *indexing* gadget on two bits. The lift of f is denoted by $f \circ \mathsf{IND}_2$. Various notions of hardness amplification on composing with the indexing gadget have been studied to give breakthrough results in communication complexity [18, 8, 9].

In this work, we use the same notion of lifting to prove lower bounds of different monomial complexity measures of *symmetric* functions. In doing so, we demonstrate the robustness of the lifting technique to prove monomial complexity lower bounds on classes of functions other than AC^0. A technical hurdle that we overcome is to show that any symmetric function can 'project' onto the lift of a suitably defined symmetric function.

1.1 Our results

In this section, we provide a detailed summary of our results.

▶ **Definition 1** (Monomial projection)**.** We call a function $g : \{-1,1\}^m \to \{-1,1\}$ a *monomial projection* of a function $f : \{-1,1\}^n \to \{-1,1\}$ if $g(x_1,\ldots,x_m) = f(M_1,\ldots,M_n)$, where each M_i is a monomial in the variables $x_1,\ldots,x_m$.

We denote the *Hamming weight* of a string $x \in \{-1,1\}^n$ to be $|x| = |\{i \in [n] : x_i = -1\}|$ (this is a natural definition since we view -1 as true, and 1 as false). For a symmetric function $f : \{-1,1\}^n \to \{-1,1\}$, define its *spectrum* or *predicate* $D_f : \{0,1,\ldots,n\} \to \{-1,1\}$ by $D_f(i) = f(x)$ where $x \in \{-1,1\}^n$ is such that $|x| = i$. Note that the spectrum (predicate) of a symmetric function is well defined. For any $f : \{-1,1\}^n \to \{-1,1\}$, define the function $f^{op} : \{-1,1\}^{3n} \to \{-1,1\}$ as follows.

$$f^{op}(x_1,\ldots,x_n,y_1,\ldots,y_n,z_1,\ldots,z_n) = f(u_1,\ldots,u_n). \tag{1}$$

where for all i, $u_i = (x_i \wedge z_i) \vee (y_i \wedge \bar{z}_i)$. Intuitively speaking, the value of z_i decides whether to feed x_i or y_i as the ith input to f. This method of lifting f was introduced by Krause and Pudlák [13].

The following lemma shows how f^{op} is a monomial projection of a symmetric function, if f was symmetric itself.

▶ **Lemma 2** (Projection Lemma). *Given a symmetric function $F : \{-1,1\}^{4n} \to \{-1,1\}$, defined by the predicate $D_F : [4n] \to \{-1,1\}$, consider the symmetric function $f : \{-1,1\}^n \to \{-1,1\}$ defined by the predicate $D_f(b) = D_F(2b+n)$ for all $b \in \{0,1,\ldots,n\}$. Then, f^{op} is a monomial projection of F.*

For any function $f : \{-1,1\}^n \to \{-1,1\}$, let $f = \sum_{S\subseteq[n]} c_S \prod_{i\in S} x_i$ be the unique multilinear expansion of f. Define the *weight* of f, denoted by $wt(f)$ to be $\sum_{S\subseteq[n]} |c_S|$.[3] The sign degree of a function f, denoted $\deg_{\pm}(f)$, is defined to be the minimum degree required by a polynomial to sign represent f on all inputs.

▶ **Definition 3** (Polynomial margin). For a polynomial of weight 1, say p, which sign represents a function f, we say that p represents f with a margin of value $\min_{x\in\{-1,1\}^n} f(x)p(x)$. Define the *polynomial margin* of f, denoted $m(f)$, as follows.

$$m(f) = \max_{p:wt(p)=1} \min_{x\in\{-1,1\}^n} f(x)p(x). \tag{2}$$

To the best of our knowledge, such a definition of the polynomial margin of a function does not appear in the past literature, although very similar notions have been studied. As we note in Theorem 24, this is a useful quantity in characterizing the weakly-unbounded error communication complexity of XOR functions.

▶ **Definition 4** (Approximate weight). Define the ϵ-approximate weight of a function $f : \{-1,1\}^n \to \{-1,1\}$, denoted by $wt_\epsilon(f)$ to be the weight of a minimum weight polynomial such that for all $x \in \{-1,1\}^n$, $|p(x) - f(x)| < \epsilon$.[4]

▶ **Definition 5** (Signed monomial complexity). The *signed monomial complexity* of a function $f : \{-1,1\}^n \to \{-1,1\}$, denoted by $\mathrm{mon}_{\pm}(f)$, is the minimum number of monomials required by a polynomial p to sign represent f on all inputs.

Note that the signed monomial complexity of a function f exactly corresponds to the minimum size Threshold of Parity circuit computing it.

Let us define a notion of error in a pointwise approximation of a function by low degree polynomials. This notion is studied widely in classical approximation theory.

Note that we do not restrict the weight of the approximating polynomial in this case.

$$\varepsilon_d(f) \triangleq \min_{p:deg(p)\leq d} \left(\max_{x\in\{-1,1\}^n} |p(x) - f(x)| \right). \tag{3}$$

▶ **Definition 6** (Approximate degree). For any function $f : \{-1,1\}^n \to \{-1,1\}$ and polynomial $p : \{-1,1\}^n \to \mathbb{R}$, we say that p approximates f uniformly to error ϵ if for all $x \in \{-1,1\}^n$, $|p(x) - f(x)| \leq \epsilon$. The ϵ-approximate degree of f, denoted $\widetilde{\deg}_\epsilon(f)$ is the minimum degree of a polynomial p which approximates f uniformly to error ϵ.

[3] Note that this notion coincides with $||\hat{f}||_1$, the spectral norm of f. However, for the purposes of this paper, we shall use the former notation.

[4] This notion coincides with the notion of the *ϵ-approximate spectral norm* of f, denoted by $||\hat{f}||_{1,\epsilon}$.

The following lemma, translates degree-hardness properties of f to monomial-hardness properties of f^{op}. The proof of this lemma is based on ideas from [13].

▶ **Lemma 7** (Lifting Lemma). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function.*
1. *If $\varepsilon_d(f) > 1 - 2^{-d}$ for some $d \geq 2$, then $m(f^{op}) \leq 2^{-c'd}$ for any constant $0 < c' < 1 - \frac{1}{d}$.*
2. $\mathrm{mon}_{\pm}(f^{op}) \geq 2^{\deg_{\pm}(f)}$. *(This part was proved in [13].)*
3. $wt_{1/3}(f^{op}) \geq 2^{c \cdot \widetilde{\deg}_{2/3}(f)}$ *for any constant $c < 1 - 3/\widetilde{\deg}_{2/3}(f)$.*

1.1.1 Applications to boolean function analysis

▶ **Definition 8.** Let $F : \{-1,1\}^n \to \{-1,1\}$ be a symmetric function. Define $r_0 = r_0(F), r_1 = r_1(F)$ to be the minimum integers such that $r_0, r_1 \leq n/2$ and $D_F(i) = D_F(i+2)$ for all $i \in [r_0, n - r_1)$. Define $r = r(F) = \max\{r_0, r_1\}$.

Using Projection Lemma, Lifting Lemma, and Paturi's theorem [17], we resolve the following conjecture by Ada et al. [1].

▶ **Theorem 9** (Conjecture 1 in [1]). *Let $F : \{-1,1\}^{4n} \to \{-1,1\}$ be any symmetric function such that $r(F) \geq 5$. Then, there exists a universal constant $c_1 > 0$ such that*

$$\log(wt_{1/3}(F)) \geq c_1 \cdot r(F).$$

One consequence of Theorem 9 is an analog of Paturi's theorem [17]. Paturi characterized the approximate degree of all symmetric functions, and we obtain a characterization of the approximate monomial complexity of symmetric functions F, in terms of $r(F)$. Let $\mathrm{mon}_{1/3}(F)$ denote the minimum number of monomials required by a polynomial to sign represent F at all points.

▶ **Theorem 10** (Approximate monomial complexity of symmetric functions). *For a symmetric function $F : \{-1,1\}^{4n} \to \{-1,1\}$,*

$$\log\left(\mathrm{mon}_{1/3}(F)\right) = \Theta^*(r(F)).$$

Theorem 10 was proved by Ada et al. [1] assuming Theorem 9. We refer the reader to [1] for a proof.

Define the *odd-even* degree of a symmetric function f, which we denote by $\deg_{oe}(f)$, to be $|i \in \{0, 1, \ldots, n-2\} : D_f(i) \neq D_f(i+2)|$.

Using Projection Lemma and Lifting Lemma, we resolve the following conjecture by Zhang [24].

▶ **Theorem 11** (Conjecture 1 in [24]). *A symmetric function $f : \{-1,1\}^n \to \{-1,1\}$ is computable by a quasi-polynomial size Threshold of Parity circuit if and only if $\deg_{oe}(f) = \log^{O(1)} n$.*

1.1.2 Applications to communication complexity of symmetric XOR functions

We consider two models of randomized communication. The first was introduced by Yao [22]. Two players, say Alice and Bob, receive a pair of inputs $x \in X$ and $y \in Y$ respectively. They want to jointly evaluate a function $F : X \times Y \to \{-1,1\}$ on the pair (x,y) by using a communication protocol that minimizes the total number of bits communicated in the worst case. The protocol is probabilistic with the requirement that $\Pr[\Pi(x,y) = F(x,y)] \geq$

$1/2 + 1/3$. The goal of the players is to design an *efficient* protocol meeting this requirement that minimizes the cost. The cost of the best protocol for computing F in this model is called its *bounded error* complexity, denoted by $R_{1/3}(F)$. We consider the bounded error complexity of XOR functions. Namely, Alice and Bob are given inputs $x, y \in \{-1,1\}^n$ and wish to compute $f(x \oplus y)$ for some given function f, where $x \oplus y$ denotes the bitwise xor of x and y.

Lee and Shraibman [14] showed that the log of the approximate weight of f is roughly a lower bound on the (quantum) bounded-error complexity of $f \circ \mathsf{XOR}$, for every f. Using our Projection Lemma and Lifting Lemma, along with this result of Lee and Shraibman [14], and an upper bound from [25] on the communication complexity of symmetric XOR functions, we obtain a characterization of the bounded error communication complexity in terms of the approximate weight of the base function.

▶ **Theorem 12.** *For any symmetric function* $f : \{-1,1\}^n \to \{-1,1\}$,

$$R_{1/3}(f \circ \mathsf{XOR}) = \Theta^*(\log wt_{1/3}(f)).$$

▶ Remark. As the method of [14] applies to even quantum communication, Theorem 12 gives even a characterization of the quantum bounded-error complexity. Recently, Zhang [23] gave upper bounds on the quantum bounded-error communication complexity of $f \circ XOR$ in terms of the log of the approximate weight of f, *provided* f has low $\mathbb{F}_2$ degree. Theorem 12 shows that when f is symmetric, then the dependence on $\mathbb{F}_2$ degree is redundant even for classical complexity.

We consider another model of randomized communication, namely the *weakly-unbounded error* model, introduced by Babai et al. [3]. A probabilistic protocol Π computes F with advantage ϵ if the probability that F and Π agree is at least $1/2 + \epsilon$ for all inputs. Denote the cost of such a protocol to be $R_\epsilon(F)$. We add a penalty term to the cost depending on the advantage, and refer to this new cost as the weakly-unbounded error complexity, or PP complexity, of F.

▶ **Definition 13.**

$$\mathsf{PP}(F) = \inf_{\epsilon} \left(R_\epsilon(F) + \log\left(\frac{1}{\epsilon}\right)\right).$$

Klauck [11] showed that the PP complexity is characterized by the well studied notion of discrepancy. Using LP duality in the full version of this paper [6], we show a general tight relationship between discrepancy of $f \circ \mathsf{XOR}$ and the margin complexity of f. Using these facts along with our Projection and Lifting Lemmas, we are able to completely characterize the PP complexity of symmetric XOR functions.

▶ **Theorem 14.** *Let* $F : \{-1,1\}^{4n} \to \{-1,1\}$ *be any symmetric function, and let* $\deg_{oe}(f) = r \geq 4$. *Then, there exists universal constants* $c, c' > 0$ *such that* $cr/\log(n/r) \leq \mathsf{PP}(F \circ \mathsf{XOR}) \leq c' r \log(n)$.

The Log Approximation Rank Conjecture [14] is the analogous version of the well-known Log Rank Conjecture, for the randomized communication complexity model. Let $\mathrm{rank}_\epsilon(M)$ denote the minimum rank of a matrix that ϵ-approximates M entry-wise. It is known that $R_{1/2-\epsilon}(F) \geq \log \mathrm{rank}_{\epsilon'}(M_F)$, where ϵ' is a constant depending on ϵ, and M_F denotes the communication matrix of F. The Log Approximation Rank Conjecture states that the lower bound is tight upto a polynomial factor.

We resolve this conjecture for symmetric XOR functions.

▶ **Theorem 15** (Log Approximation Rank Conjecture for symmetric XOR functions)**.** *Let f be any symmetric function, and $F = f \circ \mathsf{XOR}$. Then, there is a universal constant c such that*

$$\log \operatorname{rank}_{\epsilon'}(M(F)) \leq R_{1/2-\epsilon}(F) \leq \log^c \operatorname{rank}_{\epsilon'}(M(F)).$$

Ada et al. [1] prove Theorem 15 assuming Theorem 9. For a proof, and additional learning theoretic implications of Theorem 9, we refer the reader to [1].

1.2 Proof outline

First, we use an idea due to Krause and Pudlák [13], who showed that if a function f has high sign degree, f^{op} has high signed monomial complexity. We observe that their argument can be easily adapted to show a more general result. In particular, our Lemma 7 shows that the hardness of f for *low degree* polynomials, with respect to natural notions like uniform approximation and sign representation, gets amplified to corresponding hardness of f^{op} for *sparse (low weight)* polynomials. The main problem at this point is to understand the structure of f^{op}. In particular, our interest is when f^{op} suitably embeds in a symmetric function. As a first glance, symmetric functions do not seem to have the structure of a lifted function f^{op}.

At this point, inspired by the work of Krause [12], we make a simple but somewhat counter-intuitve observation that turns out to be crucial. A function g is called a monomial projection of h, if g can be obtained by substituting each input variable of h with a monomial in variables of g. What is nice about such projections is that for the polynomial sparsity measures that are relevant for us (Observation 25), the complexity of g is upper bounded by that of h. We observe (Lemma 2) that if f is a symmetric function, then there exists a symmetric function F, on a larger domain, such that f^{op} is a monomial projection of F. Moreover, the combinatorial parameters of f that caused its hardness against low-degree polynomials, nicely translate to combinatorial parameters of F that have been conjectured to cause hardness of F against sparse (low weight) polynomials.

We then find a suitable symmetric f such that f^{op} has large approximate weight and is a monomial projection of F. Lemma 2 provides such a monomial projection in which the combinatorial quantity $r(F)$ corresponds to another combinatorial quantity $\Gamma(f)$, which is defined in Section 2. Paturi's Theorem 17 shows that $\Gamma(f)$ characterizes the approximate degree of f. Our polynomial hardness amplification via Lemma 7, implies that f^{op}, and therefore F, has large approximate weight. This proves Theorem 9 which was conjectured by Ada et al. [1].

Moreover, the odd-even degree of F corresponds to the sign degree of f. Our polynomial hardness amplification via Lemma 7, implies that f^{op}, and therefore F, has large signed monomial complexity. This resolves an old conjecture of Zhang [24].

Finally, we note that F having large odd-even degree also implies that f is uniformly inapproximable by low degree polynomials. Our lifting lemma, Lemma 7 implies that f^{op}, and thus F has small polynomial margin. We then invoke a theorem, Theorem 24, proving a tight equivalence between the polynomial margin of F and the PP complexity of $F \circ \mathsf{XOR}$.

1.3 Comparison with related work

In this section, we compare our results and techniques with those of recent related works.

Shortly after our paper was put out in the public domain [6], Ada et al. [2] reported a proof of Theorem 9. However, their methods seem completely different from ours. In order to prove a lower bound on the approximate spectral norm of f, they take recourse

to the communication matrix of $f \circ \mathsf{XOR}$. Via a result of Bruck and Smolensky [4] they then show that it is enough to bound the approximate rank of this matrix. They do this by appropriately invoking a matrix theoretic lemma of Razborov [19]. This is essentially opposite to our approach. We directly prove lower bounds on the approximate spectral norm of f, using our Projection Lemma and Lifting Lemma along with Paturi's Theorem [17], without bringing communication into the picture at all. We then use the bound on the approximate spectral norm of f to prove lower bounds on the bounded error communication complexity of $f \circ \mathsf{XOR}$. It is interesting to note that, although [2] do not directly use Paturi's Theorem, the matrix theoretic lemma of Razborov that they invoke does require usage of Paturi's Theorem.

Two very recent independent works by Hatami and Qian [10] and Ada et al. [2] characterized the unbounded error communication complexity of symmetric XOR functions, strengthening our weakly-unbounded error characterization. Both the proofs involve a reduction to analyzing the unbounded error complexity of a symmetric AND function, and an invocation of a theorem of Sherstov [20] which requires heavy approximation theoretic tools. On the other hand, in order to prove our weakly-unbounded error complexity lower bound, we invoke a result of the authors [6] (a full version of this paper) relating the discrepancy of $f \circ \mathsf{XOR}$ tightly to the polynomial margin of f. Our method of lower bounding the polynomial margin of f is from first principles, and is self-contained. Although we prove a lower bound on a weaker complexity model, we shave off significant logarithmic factors from the bounds obtained by [10, 2].

In conclusion, the techniques of [10, 2] seem very specific, and use non-trivial results from [19] and [20]. Our Lifting Lemma, Lemma 7, on the other hand, applies for general functions. In particular, while the work of [10, 2] take recourse to analyzing AND functions for all their results, we build techniques that can be used directly, and in turn yield bounds on communication complexity of XOR functions. We believe our techniques will also find more use for analyzing methodically non-symmetric XOR functions, an area of active interest today. Indeed, the authors use this technique [6] to provide a simple new proof of the known separation between functions efficiently computable with weakly-unbounded error, and those efficiently computable with unbounded error, via a non-symmetric XOR function that was introduced by Goldmann et al. [7].

2 Preliminaries

We provide the necessary preliminaries in this section.

All logarithms in this paper are taken base 2.

The following result by Zhang [24] provides an upper bound on the Threshold of Parity circuit size required to compute symmetric functions.

▶ **Theorem 16** ([24]). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a symmetric boolean function such that $\deg_{oe}(f) = \log^{O(1)} n$. Then, f can be computed by a quasi-polynomial size Threshold of Parity circuit.*

The following is a result by Paturi [17] which gives us tight bounds on the approximate degree of symmetric functions.

▶ **Theorem 17** ([17]). *For any symmetric function $f : \{-1,1\}^n \to \{-1,1\}$, define the quantity $\Gamma(f) = \min\{|2k-n+1| : D_f(k) \neq D_f(k+1)$ and $0 \leq k \leq n-1\}$. Then,*

$$\widetilde{\deg}_{2/3}(f) = \Theta(\sqrt{n(n-\Gamma(f))}).$$

The following theorem was proved by Ada et al. [1], which characterizes the weight of a symmetric function.

▶ **Theorem 18** ([1]). *For any symmetric function* $f : \{-1,1\}^n \to \{-1,1\}$,

$$\log(wt(f)) = \Theta\left(r(f)\log\left(\frac{n}{r(f)}\right)\right).$$

Lee and Shraibman [14] showed that $wt_{1/3}(f)$ is a lower bound on $R_{1/3}(f \circ \mathsf{XOR})$.

▶ **Theorem 19** ([14]). *For any function* $f : \{-1,1\}^n \to \{-1,1\}$,

$$R_{1/3}(f \circ \mathsf{XOR}) = \Omega(\log wt_{1/3}(f)).$$

Shi and Zhang [25] proved that the bounded error communication complexity of symmetric XOR functions is characterized by $r(f)$.

▶ **Theorem 20** ([25]). *For any symmetric function* $f : \{-1,1\}^n \to \{-1,1\}$,

$$R_{1/3}(f \circ \mathsf{XOR}) = \Theta^*(r(f)).$$

where Θ^* *hides logarithmic factors.*

Let $\mathsf{Q}_{1/3}(F)$ denote the quantum bounded error communication complexity of F. Zhang [23] proved the following upper bound on the communication complexity of XOR functions.

▶ **Theorem 21** ([23]). *Let* $f \to \{-1,1\}^n \to \{-1,1\}$ *be any function. Suppose the* $\mathbb{F}_2$*-degree of* f *is* d*. Then,*

$$\mathsf{Q}_{1/3}(f \circ \mathsf{XOR}) = O\left(2^d\left(\log\left(wt_{1/3}(f)\right) + \log n\right)\right).$$

Define the discrepancy of a rectangle $S \times T$ under a distribution λ on $\{-1,1\}^n \times \{-1,1\}^n$ as follows.

▶ **Definition 22** (Discrepancy).

$$\mathrm{disc}_\lambda(S \times T, F) = \sum_{(x,y)\in S\times T} F(x,y)\lambda(x,y).$$

The discrepancy of F under a distribution λ is defined as

$$\mathrm{disc}_\lambda(F) = \max_{S\subseteq[n],T\subseteq[n]} \mathrm{disc}_\lambda(S \times T, F)$$

and the discrepancy of F is defined to be

$$\mathrm{disc}(F) = \min_\lambda \mathrm{disc}_\lambda(F).$$

Klauck [11] proved that the PP complexity of F is equivalent to the *discrepancy* of F.

▶ **Theorem 23** ([11]). *For any function* $F : \{-1,1\}^n \times \{-1,1\}^n \to \{-1,1\}$,

$$\mathsf{PP}(F) = \Theta\left(\log\left(\frac{1}{\mathrm{disc}(F)}\right)\right).$$

In a full version of this paper [6], the authors showed that the polynomial margin of f and the discrepancy of $f \circ \mathsf{XOR}$ are equivalent up to a constant factor.

▶ **Theorem 24** ([6]). *For any function* $f : \{-1,1\}^n \to \{-1,1\}$,

$$m(f) \le 4\mathrm{disc}(f \circ \mathsf{XOR}) \le 4m(f).$$

3 Lifting functions

In this section we first prove the Lifting Lemma, Lemma 7, which shows how certain degree-hardness properties of any function $f : \{-1,1\}^n \to \{-1,1\}$ translates to related monomial-hardness properties of f^{op}.

We then prove the Projection Lemma, Lemma 2, which shows how a symmetric function F projects onto f^{op}, for a suitably defined symmetric function f.

Finally, we list the consequences we obtain for lifting symmetric functions, which include resolving conjectures posed by Ada et al. [1], Zhang [24], and the resolution of a weak form of a conjecture by Shi and Zhang [25].

3.1 Lifting functions by the Krause-Pudlák selector

In this section, we prove the Lifting Lemma.

Proof of Lemma 7. We first prove Part 3. Suppose, to the contrary, that $wt_{1/3}(f^{op}) \leq 2^{cd}$, where c is an absolute constant, to be fixed later, and $d = \widetilde{\deg}_{2/3}(f)$. This means there exists a polynomial $p : \{-1,1\}^{3n} \to \{-1,1\}$ such that $wt(p) \leq 2^{cd}$ and $p(x)f^{op}(x) \geq 2/3$ for all $x \in \{-1,1\}^{3n}$. Say $p = \sum_{S\subseteq[3n]} w_S\chi_S$. The proof idea is to manufacture a polynomial p_2, based on p, of low *degree*, which uniformly approximates f to error $2/3$.

For this proof, we view the input variables as $\{x_{j,1}, x_{j,2}, z_j | j \in \{1,\ldots,n\}\}$, where z_j's are the 'selector' variables.

For any fixing of the z variables, define a relevant variable to be one that is 'selected' by z. Thus, for each $j \in \{1,\ldots,n\}$, exactly one of $\{x_{j,1}, x_{j,2}\}$ is relevant. Analogously, define a relevant monomial to be one that does not contain any unselected variable. For any set $S \subseteq [3n]$, define S_x to be the subset of S which contains the all the indices corresponding to the x variables.

For a uniformly random fixing of z and any subset $S \subseteq [3n]$ such that $|S_x| \geq d$,

$$\Pr_z[\chi_S \text{ is relevant}] \leq \frac{1}{2^d}.$$

$$\begin{aligned}\mathbb{E}_z[\text{wt of relevant monomials in } p\big|_z \text{ of degree } \geq d] &= \sum_{S:|S_x|\geq d} |w_S| \cdot \Pr_z[\chi_S \text{ is relevant}] \\ &\leq \frac{1}{2^d} \sum_{S:|S_x|\geq d} |w_S| \leq \frac{1}{2^d} \cdot 2^{cd}.\end{aligned}$$

Thus, there exists a fixing of the z variables such that the weight of the relevant monomials of degree at least d in $p\big|_z$ is at most $\frac{1}{2^d} \cdot 2^{cd}$. Select this fixing of z.

- Note that $p\big|_z$ is a polynomial on only the variables $\{x_{i,1}, x_{i,2} | i \in \{1,\ldots,n\}\}$. Drop the relevant monomials of degree at least d from $p\big|_z$ to obtain a polynomial p_1.
- Observe that p_1 sign represents $f^{op}\big|_z$ with error at most $\frac{1}{3} + \frac{2^{cd}}{2^d}$.
- For each $j \in \{1,\ldots,n\}$, denote the irrelevant variable by x_{j,i_j}. Consider the polynomial p_2 on n variables defined by $p_2 = \mathbb{E}_{x_{1,i_1},\ldots,x_{n,i_n}}[p_1]$, where the expectation is over each irrelevant variable being sampled uniformly and independently from $\{-1,1\}$.
- It is easy to see that any monomial containing an irrelevant variable in p_1 vanishes in p_2. Also note that p_2 is a polynomial of degree less than d, and it must sign represent f with error at most $\frac{1}{3} + \frac{2^{cd}}{2^d}$. This quantity is less than $2/3$ when $c < 1 - \frac{3}{d}$. This leads to a contradiction since we assumed that $\widetilde{\deg}_{2/3}(f) = d$.

We omit the proof of Part 1 as it follows along extremely similar lines as the proof above. ◀

3.2 Lifts as projections of symmetric functions

In this section, we prove the Projection Lemma.

The following observation is an easy consequence of definitions.

▶ **Observation 25.** *For any functions $f : \{-1,1\}^n \to \{-1,1\}$ and $g : \{-1,1\}^m \to \{-1,1\}$ such that g is a monomial projection of f, and any $\epsilon > 0$, we have*

$$\begin{aligned} \mathrm{mon}_{\pm}(g) &\leq \mathrm{mon}_{\pm}(f), \\ wt(g) &\leq wt(f), \\ wt_{\epsilon}(g) &\leq wt_{\epsilon}(f), \\ m(f) &\leq m(g). \end{aligned}$$

Proof of Lemma 2. Let $g : \{-1,1\}^{3n} \to \{-1,1\}$ be defined as follows.

$$g(x_1, \ldots x_n, y_1, \ldots, y_n, z_1, \ldots, z_n) = \\ F(x_1, \ldots, x_n, y_1, \ldots, y_n, -x_1z_1, \ldots, -x_nz_n, y_1z_1, \ldots, y_nz_n).$$

Clearly, g is a monomial projection of F. We show now that $g = f^{op}$.

For every input to g and each $i \in [n]$, define the i'th *relevant* variable to be x_i if $z_i = -1$ (define y_i to be the *irrelevant* variable in this case), and y_i if $z_1 = 1$ (x_i is irrelevant in this case). For a fixed input $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$, let b denote the number of relevant variables with value -1. Thus, there are $n - b$ irrelevant variables with value 1. Let a denote the Hamming weight of $(x_1, \ldots, x_n, y_1, \ldots, y_n, -x_1z_1, \ldots, -x_nz_n, y_1z_1, \ldots, y_nz_n)$ Then,

$$4n - 2a = \sum_{i=1}^{n} x_i + y_i - x_iz_i + y_iz_i = \sum_{i=1}^{n} x_i(1 - z_i) + y_i(1 + z_i) = 2n - 4b$$

(which is twice the sum of the values of the relevant variables)

$$\implies a = 2b + n.$$

Thus,

$$g(x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n) = D_F(2b + n) = D_f(b) \\ = f^{op}(x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n).$$

The last equality follows from Equation 1. ◀

▶ Remark. In fact, the proof of Lemma 2 implies the following.

Given a symmetric function $f : \{-1,1\}^n \to \{-1,1\}$ defined by the predicate $D_f(b)$, define a function $F : \{-1,1\}^{4n} \to \{-1,1\}$ (not necessarily symmetric) such that on inputs of Hamming weight $2b + n$, F takes the value $D_f(b)$ for all $b \in \{0, 1, \ldots, n\}$, and F takes arbitrary values on inputs of Hamming weight not in $\{2b + n : b \in \{0, 1, \ldots, n\}\}$. Then, f^{op} is a monomial projection of F.

3.3 Consequences for symmetric functions

In this section, we show consequences of hardness amplification of lifted symmetric functions.

We first prove Theorem 9.

Proof of Theorem 9. Assume that n is even and that $r-1$ is a multiple of 4. (If not, we can fix a constant number of input bits to suitable values).

Suppose $r_1(F) \geq r_0(F)$. Consider $\bar{F}$ defined by $\bar{F}(x_1,\ldots,x_n) = F(-x_1,\ldots,-x_n)$. Observe that $\log(wt_{1/3}(F)) = \log(wt_{1/3}(\bar{F}))$, and $r_0(\bar{F}) > r_1(\bar{F})$. Thus, we may assume, without loss of generality, that $r_0(F) > r_1(F)$.

Note that $D_F(r-1) \neq D_F(r+1)$. Define $F' : \{-1,1\}^{2r} \to \{-1,1\}$ by $D_{F'}(i) = D_F(i)$ for $i \in \{0,1,\ldots,2r\}$. It suffices to show $\log wt_{1/3}(F') \geq c'r$ for some universal constant $c' > 0$. Define $f : \{-1,1\}^{(r-1)/2} \to \{-1,1\}$ by $D_f(i) = D_{F'}(2i + (r-1)/2)$. By Lemma 2, f^{op} is a monomial projection of F'. Note that $D_f\left(\frac{r-1}{4}\right) \neq D_f\left(\frac{r-1}{4}+1\right)$, and thus $\Gamma(f) \leq 1$. By Theorem 17, $\widetilde{\deg}_{2/3}(f) = \Theta(r)$.

Using Part 3 of Lemma 7 and Observation 25, we obtain that there exists a universal constant $c_1 > 0$ such that

$$\log(wt_{1/3}(F)) \geq \log(wt_{1/3}(F')) \geq \log(wt_{1/3}(f^{op})) \geq c_1 r. \tag{4}$$

◀

We now prove Theorem 11, settling a conjecture of Zhang [24].

Proof of Theorem 11. Proof Idea: We define a (not too large) family of symmetric functions $\{f_i : i \in I\}$ such that f_i^{op} is a monomial projection of F for each $i \in I$. The sign degree of f_i will correspond to the number of $(k, k+2)$ sign changes in a corresponding interval of the spectrum of F. Moreover, the family $\{f_i : i \in I\}$ 'captures' all of the $(k, k+2)$ sign changes of D_F. Thus, we conclude the existence of an $i \in I$ such that the sign degree of f_i is large, and f_i^{op} is a monomial projection of F. Lemma 2, Part 2 of Lemma 7, and Observation 25 will then yield the desired result.

Assume without loss of generality, that n is a power of 3. Consider any symmetric function $F : \{-1,1\}^{4n} \to \{-1,1\}$ such that $\deg_{oe}(F) \geq 8j$ where $j \geq 2$. Suppose there were less than $4j$ many $(i, i+2)$ sign changes of D_F in $[0, 3n]$. Then, consider $\bar{F}$ defined by $\bar{F}(x_1,\ldots,x_n) = F(-x_1,\ldots,-x_n)$. Observe that $\mathrm{mon}_{\pm}(F) = \mathrm{mon}_{\pm}(\bar{F})$, and $D(\bar{F})$ has at least $4j$ many $(i, i+2)$ sign changes in $[0, 3n]$ (in particular, in $[0, n]$). Thus, we may assume, without loss of generality, that there are at least $4j$ many $(i, i+2)$ sign changes of D_F in $[0, 3n]$. Further assume that at least $2j$ of them occur when i's are even integers (if not, set one variable to -1).

Define a family $\{f_i : \{-1,1\}^{\frac{n}{3^i}} \to \{-1,1\} : i \in \{0,1,\ldots,\lceil \frac{1}{\log 3}\log(n/j)\rceil\}\}$ of symmetric functions as follows.

$$\forall b \in \left[\frac{n}{3^i}\right],\ D_{f_i}(b) = D_F\left(2b + \frac{n}{3^i}\right).$$

Note that the sign degree of f_i equals the number of $(k, k+2)$ sign changes in the spectrum of F in the interval $[\frac{n}{3^i}, \frac{n}{3^{i-1}}]$. Since D_F has at least $2j$ many $(i, i+2)$ sign changes in $[0, 3n]$, it has at least j many $(k, k+2)$ sign changes in the interval $[j, 3n]$. Thus, the spectrum of at least one of the f_i's (say f_ℓ) has at least $\frac{j}{\lceil \frac{1}{\log 3}\log(\frac{n}{j})\rceil}$ many $(k, k+1)$ sign changes (sign degree). The Projection Lemma (Lemma 2) tells us that f_ℓ^{op} is a monomial projection of F. Using Observation 25 and Part 2 of Lemma 7, we obtain that there exists a constant $c_2 > 0$ such that

$$\mathrm{mon}_{\pm}(F) \geq \mathrm{mon}_{\pm}(f_\ell^{op}) \geq 2^{\frac{c_2 j}{\log(n/j)}}.$$

The upper bound follows from Theorem 16. ◀

Next, we prove Theorem 12, providing a characterization of the bounded error communication complexity of symmetric XOR functions in terms of $wt_{1/3}(f)$.

Proof of Theorem 12. The upper bound follows from Theorem 20 and Theorem 18. The lower bound follows from Theorem 9, and Theorem 19. ◀

The proof of Theorem 14 can be found in the Appendix.

4 Conclusions

We provide a general lifting theorem, and list several applications to symmetric functions, via our Projection Lemma, including characterizations of bounded error and weakly-unbounded error communication complexity of symmetric XOR functions, characterization of the approximate weight of symmetric functions, and Threshold of Parity circuit size of symmetric functions.

Our lifting theorem applies to arbitrary functions, and we feel that it should be usable to prove lower bounds against classes of non-symmetric functions.

Zhang [23] (Theorem 21) showed an upper bound on the quantum bounded error communication complexity of $f \circ \mathsf{XOR}$ in terms of the approximate weight and the $\mathbb{F}_2$-degree of f. Theorem 12 shows that the dependence on the $\mathbb{F}_2$-degree is not required when f is symmetric, even for classical bounded error complexity. This shows the tightness of Theorem 19 for symmetric XOR functions.

We leave the reader with two open questions. Is the lower bound of Theorem 19 tight for *all* XOR functions? In particular, a positive answer would verify the Log Approximate Rank Conjecture for all XOR functions. It is well-known [16] that the real degree of a boolean function is polynomially related to its approximate degree. Does a similar relationship carry over to the spectral norm? Theorem 9 gives a positive answer for symmetric functions.

References

1 Anil Ada, Omar Fawzi, and Hamed Hatami. Spectral norm of symmetric functions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 338–349, 2012.

2 Anil Ada, Omar Fawzi, and Raghav Kulkarni. On the spectral properties of symmetric functions. *Arxiv*, 2017.

3 László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 337–347, 1986.

4 Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions, and spectral norms. *SIAM J. Comput.*, 21(1):33–42, 1992.

5 Jin-yi Cai, Frederic Green, and Thomas Thierauf. On the correlation of symmetric functions. *Mathematical Systems Theory*, 29(3):245–258, 1996.

6 Arkadev Chattopadhyay and Nikhil S. Mande. Dual polynomials and communication complexity of XOR functions. *CoRR*, abs/1704.02537, 2017. `arXiv:1704.02537`.

7 Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates VS. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

8 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1077–1088, 2015.

9 Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. *CoRR*, abs/1703.07666, 2017.

10 Hamed Hatami and Yingjie Qian. The unbounded-error communication complexity of symmetric xor functions. *Arxiv*, 2017.

11 Hartmut Klauck. Lower bounds for quantum communication complexity. *SIAM J. Comput.*, 37(1):20–46, 2007.

12 Matthias Krause. On the computational power of boolean decision lists. *Computational Complexity*, 14(4):362–375, 2006.

13 Matthias Krause and Pavel Pudlák. On the computational power of depth-2 circuits with threshold and modulo gates. *Theor. Comput. Sci.*, 174(1-2):137–156, 1997.

14 Troy Lee and Adi Shraibman. Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–398, 2009.

15 Marvin Minsky and Seymour Papert. *Perceptrons - an introduction to computational geometry*. MIT Press, 1987.

16 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994.

17 Ramamohan Paturi. On the degree of polynomials that approximate symmetric boolean functions (preliminary version). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 468–474, 1992.

18 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999.

19 Alexander A. Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya: Mathematics*, 67(1):145, 2003.

20 Alexander A. Sherstov. The unbounded-error communication complexity of symmetric functions. *Combinatorica*, 31(5):583–614, 2011.

21 Mario Szegedy. Functions with bounded symmetric communication complexity, programs over commutative monoids, and ACC. *J. Comput. Syst. Sci.*, 47(3):405–423, 1993.

22 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979.

23 Shengyu Zhang. Efficient quantum protocols for XOR functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1878–1885, 2014.

24 Zhi-Li Zhang. Complexity of symmetric functions in perceptron-like models. Master's thesis, University of Massachusetts at Amherst, 1992.

25 Zhiqiang Zhang and Yaoyun Shi. Communication complexities of symmetric XOR functions. *Quantum Information & Computation*, 9(3):255–263, 2009.

A Appendix

Proof of Theorem 14. By an extremely similar proof to that of Theorem 11 (using Part 1 of Lemma 7 instead of Part 2 of Lemma 7), it can be seen that there exists a constant $c' > 0$ such that

$$m(F) \leq \frac{1}{2^{c'r/\log(n/r)}}.$$

Thus, using Theorem 24, there exists a constant $c_1 > 0$ such that $\mathrm{disc}(F \circ \mathsf{XOR}) \leq \frac{1}{2^{c_1 r/\log(n/r)}}$. Along with Theorem 23, this proves that there exists a constant $c > 0$ such that $\mathsf{PP}(F \circ \mathsf{XOR}) \geq cr/\log(n/r)$.

We now prove the upper bound on $\mathsf{PP}(F \circ \mathsf{XOR})$. Define $S_{even} = \{i \in \{0, 2, \ldots, 4n-2\} : D_f(i) \neq D_F(i+2)\}$, and define $S_{odd} = \{i \in \{1, 3, \ldots, 4n-3\} : D_F(i) \neq D_F(i+2)\}$. By our assumption, $|S_{even}|, |S_{odd}| \leq r$.

Consider the polynomials $p_{even}, p_{odd} : \{-1, 1\}^{4n} \to \mathbb{R}$ defined by

$$p_{even}(x) = D_F(0) \cdot \prod_{i \in S_{even}} \left(4n - 2i + 1 - \left(\sum_{j=1}^{4n} x_j\right)\right)$$

and

$$p_{odd}(x) = D_F(1) \cdot \prod_{i \in S_{odd}} \left(4n - 2i + 1 - \left(\sum_{j=1}^{4n} x_j\right)\right)$$

The polynomial $p : \{-1, 1\}^{4n} \to \mathbb{R}$ defined by

$$p(x) = (1 + \chi_{[4n]}(x))p_{even}(x) + (1 - \chi_{[4n]}(x))p_{odd}(x)$$

sign represents F on $\{-1, 1\}^{4n}$.

We now use the simple observations that $wt(q_1 \cdot q_2) \leq wt(q_1) \cdot wt(q_2)$ and $wt(q_1 + q_2) \leq wt(q_1) + wt(q_2)$. Thus,

$$\begin{aligned} wt(p) &\leq 2wt(p_{even}) + 2wt(p_{odd}) \\ &\leq 2(8n)^r + 2(8n)^r \\ &\leq 4(8n)^r \end{aligned}$$

Note that all the coefficients of p are integer valued. Thus, the polynomial $p' = \frac{p}{wt(p)}$ is a polynomial of weight 1, which sign represents F with margin at least $\frac{1}{wt(p)}$. By Theorem 24 and Theorem 23,

$$\mathsf{PP}(F \circ \mathsf{XOR}) \leq O(\log(wt(p))) \leq O(r \log n).$$ ◀

Local Patterns

Joel D. Day[*1], **Pamela Fleischmann**[†2], **Florin Manea**[‡3], **and Dirk Nowotka**[§4]

1 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
jda@informatik.uni-kiel.de
2 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
fpa@informatik.uni-kiel.de
3 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
flm@informatik.uni-kiel.de
4 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
dn@informatik.uni-kiel.de

Abstract

A pattern is a word consisting of constants from an alphabet Σ of terminal symbols and variables from a set X. Given a pattern α, the decision-problem whether a given word w may be obtained by substituting the variables in α for words over Σ is called the matching problem. While this problem is, in general, NP-complete, several classes of patterns for which it can be efficiently solved are already known. We present two new classes of patterns, called k-local, and strongly-nested, and show that the respective matching problems, as well as membership can be solved efficiently for any fixed k.

1998 ACM Subject Classification F.4.3 Formal Languages, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Patterns with Variables, Local Patterns, Combinatorial Pattern Matching, Descriptive Patterns

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.24

1 Introduction

A *pattern* is a string (word) that consists of *terminal symbols* (e. g., $\mathtt{a}, \mathtt{b}, \mathtt{c}$), treated as constants, and *variables* (e. g., x_1, x_2, x_3). A pattern is mapped to a word by substituting the variables by strings of terminals. For example, $x_1x_1\mathtt{bab}x_2x_2$ can be mapped to $\mathtt{acacbabcc}$ or $\mathtt{ccbabaa}$ by the substitution $(x_1 \to \mathtt{ac}, x_2 \to \mathtt{c})$ and $(x_1 \to \mathtt{c}, x_2 \to \mathtt{a})$, respectively. If a pattern α can be mapped to a string of terminals w, we say that α matches w.

Patterns with variables appear in various areas of theoretical computer science, such as combinatorics on words (word equations [13, 20], unavoidable patterns [19]), pattern matching (generalized function matching [1, 23]), language theory (pattern languages [2]), learning

* The work of Joel Day was supported by the BMBF HPSV grant 01IH15006A.
† The work of Pamela Fleischmann was supported by the BMBF HPSV grant 01IH15006A.
‡ The work of Florin Manea was supported by the DFG grant MA 5725/1-2.
§ The work of Dirk Nowotka was supported by the DFG grant 872/3-2.

theory (inductive inference [2, 22, 24, 5], PAC-learning [14]), database theory (extended conjunctive regular path queries [3]), as well as in practice, e.g., extended regular expressions with backreferences [4, 12, 10], used in programming languages like Perl, Java, Python, etc. In most of these settings, patterns are used to express combinatorial pattern matching questions, e.g., whether a string contains certain types of regularities, encoded in the pattern. Thus, it is often necessary to solve efficiently the *matching problem*: given a pattern α and a string w, does α match w? The set of words (over a given alphabet of terminal symbols) matched by a pattern α is called the pattern language of α, denoted $L(\alpha)$. Thereby, the matching problem for inputs α and w becomes testing whether $w \in L(\alpha)$.

Unfortunately, the *matching problem* is NP-complete [2] in general. This is especially bad for some computational tasks on patterns which implicitly solve the matching problem and are therefore also intractable. For instance, in the field of algorithmic learning theory, this is the case for the task of computing *descriptive patterns* for finite sets of words [2, 7]: given a set of words S, find a pattern α that matches all words in S (i.e., $S \subseteq L(\alpha)$), and for any other pattern α' with $S \subseteq L(\alpha')$ we have that $L(\alpha') \not\subset L(\alpha)$ (in a sense, α is minimal in the set of patterns matching S, so it describes S as accurately as possible). As illustrated by Angluin in [2], descriptive patterns are useful for the inductive inference of pattern languages, which are important in the context of learning theory since they constitute a prominent example of a language class that is inferable from positive data (see, e. g., [17, 26, 28, 30, 21] and, for a survey, [29]). Moreover, descriptive patterns have also been applied in approaches of learning upper-best approximations of other types of formal languages (see [15, 11]). This, combined the other mentioned applications of pattern matching, provides good reason to identify cases in which this problem becomes tractable, and, moreover, to optimize as far as possible the corresponding algorithms.

In order to facilitate this, one looks at restricted classes of patterns. A thorough analysis [25, 28, 8, 9, 6, 27] of the complexity of the matching problem has provided some subclasses of patterns for which the matching problem is in P, when some (sometimes sophisticated) structural parameters of patterns are bounded by constants.

Our Contribution. In this paper we continue this line of work, and propose a series of new classes for which both the membership to the class and the matching problem can be decided in polynomial time. The first, k-local patterns, are a natural generalization of *non-cross patterns*, introduced in [28]. In these patterns, no occurrence of a variable $x_2 \neq x_1$ is allowed between any two occurrences of the variable x_1. For instance, $x_1\mathtt{a}\mathtt{a}x_1\mathtt{b}x_2\mathtt{a}x_2x_3x_3$ is a non-cross pattern. We can test in linear time whether a pattern is non-cross, and an efficient matching algorithm for non-cross patterns was given in [6]. The general idea behind a matching algorithm for such patterns is rather easy: for a pattern α, one can order its variables in a sequence $x_1, x_2, \ldots, x_k$ such that α can be written as $\alpha_1\alpha_2\ldots\alpha_k$ with α_i containing only occurrences of x_i and terminals for all $i \in [k]$. To match this to a word w, we first look for all ways to replace x_1 by a factor of w such that α_1 is mapped to some prefix $w[1..i_1]$ of w. Then we look for all ways to replace x_2 by a factor of w such that α_2 is mapped to a factor $w[i_1+1..i_2]$ of w with $w[1..i_1]$ being a possible image of α_1, and so on. In general, we try to find all possible ways to map $\alpha_1\cdots\alpha_j$ to a prefix $w[1..i_j]$ of w looking at how $\alpha_1\cdots\alpha_{j-1}$ were mapped to a prefix $w[1..i_{j-1}]$ of w. In the end, we check if there is a way to map α to w. This can be clearly implemented in polynomial time using dynamic programming; see [6] for an efficient algorithm based on combinatorics on words insights.

We extend this matching idea to define the class of *k-local patterns*, which are defined in full in Section 3. It is possible to match such patterns by substituting the variables with

strings in such an order that at any given point, one has matched (at most) k separate factors (blocks) of the pattern to the same number of factors of the input word. Thus, we can keep track of the k matched factors, which will be extended by assigning the next variable, in the aforementioned order. So, instead of aligning a prefix of the pattern with a prefix of the word, we align $j \leq k$ factors of the pattern to j factors of the word. For fixed k, a dynamic programming strategy will also work in this case to obtain a polynomial algorithm for matching k-local patterns, once we have the order in which the variables of the patterns are to be assigned. However, the definition of k-local patterns is much more involved than the definition of non-cross patterns and leads to a deeper understanding of the combinatorial structure of the pattern, also when compared to, for instance, patterns with a bounded number of variables or a bounded number of repeated variables. As such, it is not surprising that the problem of testing whether a pattern is k-local is a more involved computational task. Provided that k is fixed, we produce two polynomial-time algorithms. The first one decides in time $O(km^{2k})$ whether a pattern of length m is k-local. In this process we can construct the order in which the variables are to be assigned and we derive a second algorithm that, given a pattern α of length m and a word w of length $n \geq m$, decides whether α is k-local and, if so, whether α matches w in time $O(mkn^{3k-1})$.

While the definition of k-local patterns is somehow algorithmic, it does not say much about the syntactic structure of these patterns. We give precise structural characterizations of the 1- and 2-local patterns, and show that in the case of 1-local patterns this leads to a linear algorithm deciding whether a pattern is 1-local (Theorem 10) and an $O(mn^2 \log n)$-time algorithm matching a 1-local pattern of length m to a word of length n (Theorem 11).

These algorithms are not a direct implementation of the straightforward dynamic programming approach, but rather they combine this with some insights in the structure of k-local patterns and use non-trivial string processing data structures. Moreover, while k-local patterns (for fixed k) can be shown also to have bounded treewidth, and hence the techniques from [25] may be used to derive a polynomial time matching algorithm, it is worth noting that both for general k, and especially in the case $k = 1$, the algorithms presented in the current paper provide a significant improvement in complexity.

The palindromic structure of 1-local patterns (see Lemma 9) also gives rise to the idea of separating parts of a pattern with parenthesizing variables. We define the class of *strongly-nested patterns* (Definition 19, Section 5), which are a restriction on the nested patterns, and more generally, the mildly entwined patterns introduced in [25]. This further restriction comes with advantages: we show that one can decide whether a pattern is nested in linear time, and that one can match a pattern of length m to a word of length n in $O(mn^3)$ time, also a significant improvement on the $O(mn^6)$ algorithm known for mildly entwined patterns.

Additionally, we show that there is no constant k such that all strongly-nested patterns are k-local. In particular, we construct a simple algorithm based on combinatorial insights which computes, for a nested pattern α, the minimum ℓ such that α is ℓ-local. It is straightforward to infer from the algorithm that there are nested patterns of length m that are $\Theta(\log m)$-local.

Shinohara Classes. In addition to considering restricted classes of patterns, one can restrict the concept of descriptiveness to any given class Π of patterns: a pattern α is Π-descriptive if $\alpha \in \Pi$, $S \subseteq L(\alpha)$ and there is no other pattern $\beta \in \Pi$ with $S \subseteq L(\beta) \subset L(\alpha)$. Based on this idea, in [28], Shinohara initiated a line of research by providing a polynomial-time algorithm, for a (very) restricted class of patterns and a finite set of strings. This approach was extended in [7] to the notion of a *Shinohara-class* of patterns: a class of patterns Π is a *Shinohara-class* if it contains the set $\{x_1x_2 \cdots x_k \mid k \in \mathbb{N}\}$ and, for every $\alpha \in \Pi$, the pattern

α' obtained by substituting some length i suffix of α by a sequence of new variables $y_1 y_2 \cdots y_i$ is also in Π. Within the set of Shinohara-classes of patterns, it was shown that Π-descriptive patterns can be computed in polynomial time if and only if the question whether $\alpha \in \Pi$ and the matching problem for Π are polynomial-time decidable. Hence, it is interesting to identify Shinohara classes for which the matching problem can be solved efficiently. It is easily seen that both k-local patterns and strongly-nested patterns are Shinohara classes (provided $k \geq 1$). Thus, we can conclude that, provided k is treated as constant, descriptive patterns can be computed for both these classes in polynomial time.

2 Basic Definitions

For detailed definitions regarding combinatorics on words we refer to [18], [19]. For $n, i, j \in \mathbb{N}_0$ with $i \leq j$, let $[n] = \{1, \ldots, n\}$ and $[i, j] = \{i, i+1, \ldots, j-1, j\}$. In this paper, $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ denotes a finite alphabet of *terminal symbols* and $X = \{x_1, x_2, \ldots\}$ a potentially infinite alphabet of *variables*. We assume $\Sigma \cap X = \emptyset$. Words in $(X \cup \Sigma)^*$ are *patterns*, while words in Σ^* are *terminal words* (usually just words). Patterns in X^* are called *terminal-free*. We denote the set of patterns by $\textsc{Pat}_\Sigma = (X \cup \Sigma)^*$ and define $\textsc{Pat} = \bigcup_\Sigma \textsc{Pat}_\Sigma$. The *empty word* is denoted by ε and the *length* of a word w by $|w|$. Given a pattern α, let $\mathrm{alph}(\alpha)$ and $\mathrm{var}(\alpha)$ be respectively the smallest sets $\Delta \subseteq \Sigma$ and $Y \subseteq X$ such that $\alpha \in (\Delta \cup Y)^*$. Given a word $w = \mathtt{a}_1 \mathtt{a}_2 \ldots \mathtt{a}_n$, the reversal w^R of w is $\mathtt{a}_n \mathtt{a}_{n-1} \ldots \mathtt{a}_1$. For $w \in \Sigma^*$ and each $i, j \in [|w|]$ with $i \leq j$, let $w[i..j] = w[i] \cdots w[j]$, where $w[k]$ represents the k^{th} letter of w for $k \in [|w|]$. Each word $w[i..j]$ is a *factor* of w. If $0 < |w[i..j]| < |w|$ then $w[i..j]$ is a *proper factor* of w. We use the term 'blocks' for factors which satisfy a property but which are not contained strictly within a larger factor satisfying the same property. Given a variable x and pattern α, a *block* of x is a factor $\beta = \alpha[i..j]$ with $\mathrm{var}(\beta) = \{x\}$ such that either $i = 1$ or $\alpha[i-1] = y_1 \neq x$ and either $j = |\alpha|$ or $\alpha[j+1] = y_2 \neq x$. *Marked blocks* are defined similarly in Section 3. Given a variable x, we denote a block of x by $[x]^\mathtt{b}$ (resp. $x^\mathtt{b}$ if it necessarily contains at least one x). Using this notation we can define classes of patterns, e.g., $\alpha \in [x]^\mathtt{b} y z^\mathtt{b}$ implies that α has the form given by the regular expression $(\Sigma \cup \{x\})^* y (\Sigma \cup \{z\})^+$.

A *substitution* (*for* α) is a mapping $h : \mathrm{var}(\alpha) \to \Sigma^+$. For every $x \in \mathrm{var}(\alpha)$, we say that x *is substituted by* $h(x)$. The word obtained by substituting every occurrence of a variable x in α by $h(x)$ and leaving the terminals unchanged is denoted by $h(\alpha)$. For instance, we consider the pattern $\beta = x_1 \mathtt{a} x_2 \mathtt{b} x_2$ and the words $u = \mathtt{bacabca}$, $v = \mathtt{aaaabaa}$. It can be verified that $h(\beta) = u$, where $h(x_1) = \mathtt{b}$, $h(x_2) = \mathtt{ca}$ and $g(\beta) = v$, where $g(x_1) = \mathtt{a}$ and $g(x_2) = \mathtt{aa}$. Given a pattern α, the set $\{h(\alpha) \mid h \text{ is a substitution}\}$ is the *pattern language* of α, denoted $L(\alpha)$. The *matching problem*, denoted by Match, is to decide for a given pattern α and word w, whether there exists a substitution h with $h(\alpha) = w$.[1] For any $P \subseteq \textsc{Pat}$, the *matching problem for* P is to decide for a given pattern $\alpha \in P$ and word w, whether there exists a substitution h with $h(\alpha) = w$.

A pattern α is *regular* if each variable $x \in X$ occurs at most once. Given a pattern α and $y \in \mathrm{var}(\alpha)$, the *scope of* y *in* α is defined by $\mathrm{sc}_\alpha(y) = [i, j]$, where i is the leftmost and j the rightmost occurrence of y in α. The scopes of some variables $y_1, y_2, \ldots, y_k \in \mathrm{var}(\alpha)$ *coincide in* α if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. We denote the *scope coincidence degree* (scd for short) of α by $\mathrm{scd}(\alpha)$, which is the maximum number of variables in α such that their scopes coincide. For example, the scopes of all variables coincide in $\alpha_1 = x_1 x_2 x_1 x_2 x_3 x_1 x_2 x_3$, but the scopes of

[1] There exist variants of the matching problem where substitutions can also *erase* variables by mapping them to ε. Here we only consider non-erasing substitutions.

x_1 and x_3 do not coincide in $\alpha_2 = x_1x_2x_1x_2x_3x_2x_3x_3$; thus, $\mathrm{scd}(\alpha_1) = 3$ and $\mathrm{scd}(\alpha_2) = 2$. For every $k \in \mathbb{N}$, let $\mathrm{PAT}_{\mathrm{scd}\leq k}$ denote the set of patterns α with $\mathrm{scd}(\alpha) \leq k$. The class of *non-cross* patterns (see [28])) coincides exactly with $\mathrm{PAT}_{\mathrm{scd}\leq 1}$. A set $\Pi \subseteq (X \cup \Sigma)^*$ is a *Shinohara class* if $\{x_1x_2 \cdots x_k \mid k \in \mathbb{N}\} \subseteq \Pi$ and, for every $\alpha \in \Pi$ and $i \in [|\alpha|]$, we have $\alpha' = \alpha[1..i]y_1y_2 \ldots y_{|\alpha|-i} \in \Pi$, where $y_1, y_2, \ldots, y_{|\alpha|-i} \in X \setminus \mathrm{var}(\alpha)$ with $y_j \neq y_k$ for $1 \leq j < k \leq |\alpha|-i$. For a class Π of patterns and a finite set of words $S \subset \Sigma^*$, a pattern $\alpha \in \Pi$ is $\Pi-$*descriptive* of S if there does not exist a pattern $\beta \in \Pi$ such that $S \subseteq L(\beta) \subset L(\alpha)$.

3 k-Local Patterns

In the following section, the main ideas surrounding k-locality are presented along with the formal definition and some initial observations. At the end of the section, two of the main results motivating the idea of k-locality are presented: namely that if k is a fixed constant, then the membership and matching problems may be solved efficiently for k-local patterns.

Intuitively, the notion of k-locality involves marking the variables in the pattern in some arbitrary order until all the variables are marked. The pattern is k-local if this marking can be done while never creating more than k marked blocks. Variables which only occur adjacent to those which are already marked can be marked "for free" – without creating any new blocks, and thus a valid marking sequence allows a sort-of parsing of the pattern whilst maintaining a degree of closeness (locality) to the parts already parsed.

As with various other classes of patterns motivated by efficient matching algorithms (bounded scd, non-cross, etc.), k-locality deals only with the relative positions of variables, while the terminal symbols are not taken into account. Hence, before we introduce k-locality formally, it is convenient to consider the underlying pattern consisting only of variables – the *skeleton*. For example, the skeleton of $\beta = \mathtt{a}\mathtt{a}xx\mathtt{c}y\mathtt{b}\mathtt{a}z\mathtt{a}y\mathtt{a}y$ would be $\alpha = xxyzyy$.

▶ **Definition 1.** Let $\beta \in (X \cup \Sigma)^*$. The *skeleton* of β is the (unique) pattern $\alpha = y_1 \ldots y_n$ with $y_i \in X$ for $i \in [n], n \in \mathbb{N}$, such that there exist words $a_0, \ldots, a_n \in \Sigma^*$ with $\beta = a_0y_1a_1 \ldots a_{n-1}y_na_n$.

Next, the idea of marking a variable is formalized. For each variable $x \in X$, we produce a marked version $\overline{x}$. Marking x corresponds to substituting *every* occurrence of x in a pattern with its marked equivalent $\overline{x}$.

▶ **Definition 2.** Let $\overline{X} = \{\overline{x} \mid x \in X\}$ be the set of *marked variables* (with $\overline{X} \cap X = \emptyset$). For the skeleton α of a pattern $\beta \in (X \cup \Sigma)^*$, a *marking sequence* of the variables occurring in β, is an enumeration $x_1, x_2, \ldots, x_{|\mathrm{var}(\beta)|}$ of $\mathrm{var}(\beta)$. A variable x_i is called *marked at point* $k \in \mathbb{N}$ (both in β and α) if $i \leq k$. Moreover, we define α_k, *the marked skeleton of* β *at point* k, as the string obtained from α by replacing all x_i with $i \leq k$ by $\overline{x_i}$. A factor of α_k is a *marked block* if it consists of one or more marked variables and is maximal in the sense that it is not contained within another such factor.

Using the idea of a marking sequence, we can now define the k-locality of a pattern.

▶ **Definition 3.** A pattern $\beta \in (X \cup \Sigma)^*$, with skeleton α, is *k-local* for $k \in \mathbb{N}_0$ if there exists a marking sequence $x_1, \ldots, x_\ell$ of $\mathrm{var}(\beta)$, such that, for all $i \leq \ell$ we have that α_i, the marked skeleton of β at point i, has at most k marked blocks. A pattern is called *strictly* k-local if it is k-local but not $(k-1)$-local. Let $\mathrm{PAT}_{k\text{-loc}}$ denote the class of k-local patterns.

Consider the pattern $\beta = \mathtt{a}x\mathtt{a}yx\mathtt{b}y\mathtt{a}z\mathtt{a}\mathtt{b}xz$, whose skeleton is $\alpha = xyxyzxz$. If we consider the marking sequence y, x, z, then we obtain the following (partially) marked patterns:

$$\alpha_1 = x\ \overline{y}\ x\ \overline{y}\ z\ x\ z, \quad \alpha_2 = \overline{x\ y\ x\ y}\ z\ \overline{x}\ z, \quad \alpha_3 = \overline{x\ y\ x\ y\ z\ x\ z}.$$

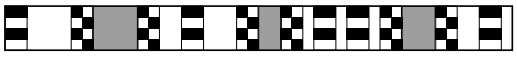


Figure 1 Lemma 5: if i blocks are already marked (grey areas), the next variable in the marking sequence may appear at $2i$ positions next to the marked blocks (chessboard), and at $k-2i$ positions not connected to marked blocks (striped).

Note that the final pattern (in this case α_3) will always be completely marked, and hence has exactly one marked block. However, since α_1 and α_2 both have exactly two marked blocks, β (and α) are 2-local. On the other hand, they are not 1-local since every alternative marking results in at least two blocks: the other possibilities for α_1 are $\overline{x}\,y\,\overline{x}\,y\,z\,\overline{x}\,z$ and $x\,y\,x\,y\,\overline{z}\,x\,\overline{z}$. The particular cases of 1- and 2-local patterns are considered in more detail in Section 4 where structural characterizations are given.

Before moving on to the first main results regarding the membership and matching problems, a few basic observations for k-local patterns are presented. Firstly the relationship between a k-local pattern and its factors is considered. Lemma 5 is illustrated by Figure 1.

▶ **Lemma 4.** *Let $\beta \in (X \cup \Sigma)^*$ be k-local. Then every factor of β is k-local. Moreover, if β is strictly k-local and $k \geq 2$, then there exists a proper factor of β which is not $(k-2)$-local.*

▶ **Lemma 5.** *Let $\beta \in (X \cup \Sigma)^*$ be a strictly k-local pattern with skeleton α. For all $x \in \mathrm{var}(\beta)$, the number of blocks $x^{\mathfrak{b}}$ occurring in α is at most $2k$. Moreover, there exists a variable $y \in \mathrm{var}(\beta)$ such that the number of blocks $y^{\mathfrak{b}}$ occurring in α is at most k.*

With regards to existing classes of patterns for which the matching problem may be efficiently solved, k-local patterns generalize the non-cross patterns in a significant manner. In fact, it follows from the results in Section 4 that non-cross (and therefore regular) patterns are 1-local. On the other hand, it can be seen with a little effort that for fixed k, k-local patterns also have treewidth bounded by $2k$. Thus are a more restricted subclass of the patterns considered in [25]. Furthermore, k-locality is incomparable to the notion of bounded scope coincidence degree, as witnessed by the following examples. The pattern $(xy)^{k+1}$ has scope coincidence degree one, while it is strictly $(k+1)$-local: marking either x or y first will result in exactly $k+1$ marked blocks. On the other hand, the pattern $x_1x_2\ldots x_{n-1}x_nx_{n-1}\ldots x_2x_1$ is 1-local (simply mark the x_is in decreasing order), but has scope coincidence degree n provided $x_i \neq x_j$ for all $i, j \in [n]$. Note also the special degenerate case of 0-local patterns.

▶ Remark 6. Let $\beta \in (X \cup \Sigma)^*$ be a pattern. Then β is 0-local if and only if $\beta \in \Sigma^*$.

Finally, the main results of this section are given. Theorems 7 and 8 show that, if k is fixed, it is possible to decide whether a pattern is k-local and also to match a k-local pattern to a word in polynomial time. The degree of the polynomial, however, depends on k. In particular, Theorem 8 provides an improvement when compared to patterns with bounded treewidth in general.

▶ **Theorem 7.** *Given a pattern $\beta \in (X \cup \Sigma)^*$ of length m, we can decide in $O(m^{2k}k)$ time whether $\beta \in \mathrm{PAT}_{\text{k-loc}}$. If the answer is positive, we can produce in the same time a marking sequence witnessing that β is k-local.*

The algorithm deciding whether a pattern is k-local keeps track of all possibilities having $j \leq k$ marked blocks in the skeleton of β, after exactly i variables were marked. Then, for each possibility, a new (the $(i+1)^{th}$) variable is selected, its (at most $2k$, see Lemma 5) blocks are marked, and they are merged with the already marked blocks. If the resulting skeleton has at most k marked blocks, it is saved and will be processed later, when the

next variable has to be marked. The pattern is k-local if and only if its entire skeleton can be marked in this way. To get the stated running time, one has to be careful when a new variable is selected: if we already have k marked blocks in our skeleton (and there are $O(m^{2k})$ possibilities), then the new variable has to be adjacent to one of them. If there are less than k marked blocks, then we are not so restricted on how to choose it; however, in this case, the number of skeletons with $j < k$ marked blocks that we consider is lower, only $O(m^{2k-2})$.

We are also able to take advantage of the k-locality structure to solve the matching problem more efficiently than the more general (but less direct) approach given in [25].

▶ **Theorem 8.** MATCH *for* $\text{PAT}_{\text{k-loc}}$ *can be decided in* $O(mkn^{\max(3k-1,2k+1)})$ *time, where* m *is the length of the input pattern and* n *is the length of the input word.*

To solve the matching problem for $\text{PAT}_{k\text{-loc}}$ we use essentially the same idea as above. We now have the order in which the variables have to be assigned, so also the marked factors in the pattern, but we need to keep track to which factors of the input word they correspond. Then we try to assign every new variable so that it fits nicely around the already matched factors. This is done efficiently using a data structure from [16]: given a word w and a one-variable pattern γ (so, $|\operatorname{var}(\gamma)| = 1$), one can produce a compact representation of all the g factors of w matching γ in $O(|\gamma||w|)$ time; moreover, we can obtain all the g factors of w matching γ in $O(|g|)$ time. This allows us to test efficiently which factors of w match any of the one-variable blocks of β, and, ultimately, to assign a value to each variable.

In comparison to the algorithm from [25] for patterns of bounded treewidth, which firstly constructs relational structures (which may be thought of as graphs) from α and w, and solves the homomorphism problem on these relational structures, the above algorithm exploits directly the locality structure present in the patterns. The advantage of this more focussed approach is that it allows for a considerable improvement in the required time, reducing the exponent of n from $4k + 4$ to $3k - 1$.

In [8, 9] it was shown that for many choices of numerical parameters, the matching problem is $W[1]$-hard. In particular, the problem is $W[1]$-hard when parameterized by the number of variables and the maximum number of occurrences of a variable. Since these values bound the total number of variables (i.e., the length of the skeleton), they also bound the strict k-locality of a pattern. Consequently, the matching problem is also $W[1]$-hard when parameterized by strict k-locality. In addition to this observation, we add the following two conjectures: deciding whether a pattern is k-local, when given k as input together with the pattern, is NP-complete; and thus that computing the minimum k such that a pattern is k-local is a computationally hard problem as well. Finally, we conjecture that the problem of deciding whether a pattern is k-local, parametrized in k, is also $W[1]$-hard.

4 1-Local and 2-Local Patterns

For 0-local patterns, a straightforward structural characterization exists: they are precisely the patterns without variables. The general case, however, is considerably more complex. Nevertheless, for small k (1 and 2), it is possible to give some recursive, structural characterizations. Firstly, given a 1-local pattern α, and a variable $x \notin \operatorname{var}(\alpha)$, we can add occurrences of x to the start of α, the end, or both, while preserving 1-locality. Moreover, this operation is sufficient to characterize 1-local patterns recursively. Since k-locality depends only on the skeleton of a pattern, in the following lemma, only terminal-free patterns are considered.

▶ **Lemma 9.** *A terminal-free pattern* $\alpha \in X^*$ *is* 1*-local if and only if* α *is empty or there exist a shorter, 1-local pattern* α' *and a variable* $x \notin \operatorname{var}(\alpha')$ *such that* $\alpha \in [x]^\flat \alpha' [x]^\flat$ *holds.*

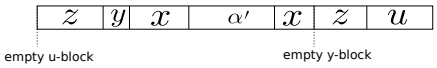


Figure 2 Extending a 1-local structure at both sides simultaneously by a possibly empty block of the same variable preserves the 1-locality.

It follows from this characterization that the structure of 1-local patterns has at its core a sort-of palindromic structure. In particular, if each new variable x is added to both sides of α, then, ignoring the number of repetitions, we get a palindrome in the variables. More generally, if a new variable x is added to only one side of α, then this may be viewed as adding a zero-repetition of x to the other, and hence it is still possible to infer an underlying palindromic structure (Figure 2). This structure, along with Lemma 9 is useful because it leads to more efficient membership and matching algorithms for 1-local patterns.

▶ **Theorem 10.** *Given a pattern $\beta \in (X \cup \Sigma)^*$ of length m, we can decide in $O(m)$ time whether $\beta \in \text{PAT}_{\text{1-loc}}$. If the answer is positive, we can produce in the same time a marking sequence witnessing that β is 1-local.*

While the theorem above follows immediately, as only a stack is needed to check that the variables occurring in more than one block in the skeleton of β form this palindromic structure, solving the matching problem efficiently requires a much finer combinatorial analysis of the way we can assign values to the variables of the pattern β. Intuitively, matching a 1-local pattern β, of length m, to a word w, of length n, is done as follows. We first get the marking sequence witnessing the 1-locality of β and then start assigning values to its variables, in the order indicated by this sequence. Doing this, after trying to assign the first s variables, we identify all possibilities to match the factor β' of β, which contains all the occurrences of the s assigned variables, to factors $w[i..j]$ of w. Now, if the next variable to be assigned is x, then the blocks of this variable x must match factors adjacent to $w[i..j]$. Our algorithm has now two phases. First, for each pair i, j such that $w[i..j]$ was a match for β', we identify in $O(\log n)$ time some basic ways to assign the variable x, as described above. This gives us some new factors $w[i'..j']$ that match β'', the factor of β that includes β' and all occurrences of x. In overall $O(n^2 \log n)$ time, we can extend these factors using their combinatorial properties (for instance, periodicity) to find all pairs i'', j'' such that $w[i''..j'']$ matches β''. We extend, thus, the matchings constructed by assigning one variable at a time, in the order of the marking sequence, just as we did in the algorithm of Theorem 8, but this time we do it more efficiently using combinatorics on words insights.

▶ **Theorem 11.** MATCH *for* $\text{PAT}_{\text{1-loc}}$ *can be decided in $O(mn^2 \log n)$ time, where m is the length of the input pattern and n is the length of the input word.*

The "palindromic" structure present in 1-local patterns, while simple, is also an integral part of understanding patterns with larger values k, since it describes precisely when additional variables may be marked without creating new marked blocks and hence the idea of locality. The characterization given in the lemma rests on the fact that, when marking a 1-local pattern, at any point we have a single marked block. Since two marked blocks are not allowed, it is then only possible to mark variables which only occur directly to either side of the current block. In the more general case, we may have a number of blocks which are not 1-local, but rather k-local for some values k. Then, using the same idea, it is possible to continue to mark variables only occurring adjacent to each of the blocks in order, and we get the same palindromic structure. Formally, this generalization is as follows.

▶ **Definition 12.** Patterns $\gamma_1, \gamma_2, \ldots, \gamma_n \in (X \cup \Sigma)^*$ are *homogeneously ordered* if for every $x, y \in X$, $x \neq y$ such that $\gamma_i = \delta_1 x \delta_2 y \delta_3$ for some $i \in [n]$ and $\delta_1, \delta_2, \delta_3 \in (X \cup \Sigma)^*$, there does not exist $j \in [n]$ such that $\gamma_j = \delta_1' y \delta_2' x \delta_3'$ where $\delta_1', \delta_2', \delta_3' \in (X \cup \Sigma)^*$. A pair of patterns (γ_1, γ_2) is called *mutually-palindromic* if γ_1 and γ_2^R are homogeneously ordered.

Essentially, patterns are homogeneously ordered if they are non-cross, and the variables appear in the same order from left to right in every pattern (though some variables may not appear in every pattern). Pairs in which one pattern is reversed provide the necessary outward palindromic structure. For example, the patterns $xyzw, xw, yw$ and w are homogeneously ordered, while the patterns $xxyzz$ and $zxyy$ are not. Similarly, the pair $(xxyzzz, zzxx)$ is mutually-palindromic, since $xxzz$ and $xxyzzz$ are homogeneously ordered.

▶ Remark 13. Lemma 9 implies that for every non-empty 1-local pattern $\alpha \in (X \cup \Sigma)^*$ there exists a (not necessarily unique) mutually-palindromic pair (γ_1, γ_2) such that $\alpha = \gamma_1\gamma_2$. Moreover, concatenating a mutually-palindromic pair always gives a 1-local pattern.

A more general version of this observation demonstrates how mutually-palindromic patterns allow for blocks (or patterns) to be extended without increasing the k-locality.

▶ **Lemma 14.** *Let $\alpha, \beta_1, \beta_2 \in (X \cup \Sigma)^*$ such that (β_1, β_2) is a mutually-palindromic pair and $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta_1\beta_2) = \emptyset$. Then $\beta_1\alpha\beta_2$ is k-local if and only if α is k-local, for all $k \in \mathbb{N}$.*

Before giving the characterization of 2-local patterns, which, as is to be expected, is more involved than for 1-local patterns, the idea of 2-local pairs is considered.

▶ **Definition 15.** Let $\alpha_1, \alpha_2 \in (X \cup \Sigma)^*$. Then (α_1, α_2) is a 2-*local pair* if there exists a marking sequence of the variables in $\mathrm{var}(\alpha_1) \cup \mathrm{var}(\alpha_2)$ such that when applied simultaneously to mark α_1 and α_2, the sum of the number of blocks in the marked skeletons at every point is at most two.

For example, $(xyxyz, x)$ is a 2-local pair, due to the marking sequence y, z, x, while $(xyxy, xy)$ is not, since whichever of x, y is marked first, there will be a total of three marked blocks (two blocks in the first pattern and one in the second). Moreover, if (α_1, α_2) is a 2-local pair, then for a pattern $\beta_1\alpha_1\beta_2\alpha_2\beta_3$ such that $\mathrm{var}(\alpha_1\alpha_2) \cap \mathrm{var}(\beta_1\beta_2\beta_3) = \emptyset$, there exists a marking sequence which marks all the variables in α_1 and α_2 using at most two marked blocks. If β_2, α_1 and α_2 are non-empty, then the marking sequence requires exactly two marked blocks. A consequence is the following simple characterization of 2-local patterns.

▶ **Lemma 16.** *A pattern $\alpha \in (X \cup \Sigma)^*$ is 2-local if and only if there exist homogeneously ordered patterns $\beta_1, \beta_2, \beta_3, \beta_4 \in (X \cup \Sigma)^*$ and a 2-local pair $(\alpha', \alpha'') \in ((X \cup \Sigma)^*)^2$ such that $\alpha = \beta_1\alpha'\beta_2^R\beta_3\alpha''\beta_4^R$, where $\mathrm{var}(\beta_1\beta_2\beta_3\beta_4) \cap \mathrm{var}(\alpha'\alpha'') = \emptyset$, and $|\alpha'\alpha''| < |\alpha|$.*

Nevertheless, the characterization still says little about the structure of 2-local patterns due to the fact that the structure of 2-local pairs is not discussed. A recursive characterization, conceptually similar to Lemma 9 is given for 2-local pairs below.

▶ **Lemma 17.** *Let $\alpha, \beta \in (X \cup \Sigma)^*$ be patterns such that $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) \neq \emptyset$. Then (α, β) is a 2-local pair if and only if there exists a 2-local pair $\alpha', \beta' \in (X \cup \Sigma)^*$, homogeneously ordered patterns $\gamma_1, \gamma_2, \gamma_3, \gamma_4 \in (X \cup \Sigma)^*$ and a variable $x \in X$ such that either*

- *$\alpha \in \gamma_1[x]^\flat\alpha'[x]^\flat\beta'[x]^\flat\gamma_2^R$ and $\beta \in \gamma_3[x]^\flat\gamma_4^R$, or*
- *$\alpha \in \gamma_1[x]^\flat\gamma_2^R$ and $\beta \in \gamma_3[x]^\flat\alpha'[x]^\flat\beta'[x]^\flat\gamma_4^R$.*

where $\mathrm{var}(\gamma_1\gamma_2\gamma_3\gamma_4), \{x\}$, and $\mathrm{var}(\alpha'\beta')$ are pairwise disjoint.

▶ **Lemma 18.** *Let $\alpha, \beta \in (X \cup \Sigma)^*$ be patterns with $\mathrm{var}(\alpha) \cap \mathrm{var}(\beta) = \emptyset$. Then (α, β) is a 2-local pair if and only if both α and β are 2-local, and at most one of α, β is strictly 2-local.*

Using Lemmas 16, 17 and 18, it is possible to derive a dynamic programming algorithm for recognizing 2-local patterns. However, it is worth pointing out that such an algorithm runs in the same time as the one given by Theorem 7 and hence, even in the case of 2-locality, it is reasonable to expect that any improvements would require significant effort.

5 Strongly-Nested Patterns

In the current section, another class of patterns which satisfy certain locality-inspired structural constraints are considered, namely the *strongly-nested patterns*, which form a subclass of the nested patterns, and consequently the mildly entwined patterns defined in [25]. It is shown that the membership problem for the class of strongly-nested patterns can be solved in linear time, and that the matching problem can be solved in $O(mn^3)$ time, where m is the length of the pattern and n is the length of the word to be matched: a considerable improvement on the mildly entwined patterns in general, for which the state of the art algorithm requires $O(mn^6)$ time. It is then shown that being strongly-nested is orthogonal to k-locality for fixed k, and that the optimal k – the smallest such that a pattern is k-local – can be computed efficiently for this class. The definition of strongly-nested patterns is given formally as follows.

▶ **Definition 19.** A pattern $\alpha \in (X \cup \Sigma)^*$ is *strongly-nested* if, for every variable $x \in \mathrm{var}(\alpha)$ there exist $\alpha_1, \alpha_2, \alpha_3 \in X^*$ such that $\alpha \in \alpha_1[x]^{\flat}\alpha_2[x]^{\flat}\alpha_3$, where $\{x\}$, $\mathrm{var}(\alpha_1\alpha_3)$ and $\mathrm{var}(\alpha_2)$ are pairwise disjoint. The set of all strongly-nested patterns is denoted $\textsc{Pat}_{\mathrm{nest}}$.

An alternative, inductive definition for strongly-nested patterns is the following: Patterns β with $|\mathrm{var}(\beta)| = 1$ are basic strongly-nested patterns. If β_1, β_2 are variable-disjoint strongly-nested patterns, x is a variable not contained in β_1, and $\gamma_1, \gamma_2 \in (\Sigma \cup \{x\})^*$, then both $\beta_1\beta_2$ and $\gamma_1\beta_1\gamma_2$ are strongly-nested patterns. Essentially, the construction of $\gamma_1\beta_1\gamma_2$ corresponds to wrapping blocks of x around the skeleton of β_1, which suggests the idea of nesting.

The alternative definition of strongly-nested patterns suggests also a notion of *depth* of such a pattern. To begin with, we say that the $\mathrm{depth}(\beta) = 1$ if $|\mathrm{var}(\beta)| = 1$. Further, if β_1, β_2 are variable-disjoint strongly-nested patterns, then $\mathrm{depth}(\beta_1\beta_2) = \max(\mathrm{depth}(\beta_1), \mathrm{depth}(\beta_2))$. Finally, if x is a variable not contained in β_1, and $\gamma_1, \gamma_2 \in (\Sigma \cup \{x\})^*$ with $x \in \mathrm{var}(\gamma_1) \cap \mathrm{var}(\gamma_2)$, then $\mathrm{depth}(\gamma_1\beta_1\gamma_2) = 1 + \mathrm{depth}(\beta_1)$. It is a simple consequence that $\mathrm{scd}(\beta) = \mathrm{depth}(\beta)$, if β is a strongly-nested pattern.

▶ **Theorem 20.** *For a pattern $\beta \in (X \cup \Sigma)^*$, of length m, we can decide in $O(m)$ time whether $\beta \in \textsc{Pat}_{\mathrm{nest}}$.*

As for 1-local patterns, the theorem above follows easily, as we can use a stack to check that the variables which occur in more than one block in the skeleton of β form a correct strongly-nested structure (i.e., they do not interleave like $..x..y..x..y..$ and occur in at most two blocks). In fact, we can represent the strongly-nested structure of a pattern as follows: let $\gamma \in (X \cup \Sigma)^*$ be a strongly-nested pattern that starts with a variable. We associate to γ a binary tree T_γ defined as follows:

1. If $|\mathrm{var}(\gamma)| = 1$, then T_γ has a single node, labelled with γ.
2. If $\gamma = \gamma'\gamma''$ where γ' and γ'' are variable disjoint strongly-nested patterns, both starting with a variable, then the tree associated with γ consists of a node labelled with $\odot$ which has two children. The left child is the tree $T_{\gamma'}$, and the right child is the tree $T_{\gamma''}$. If there are multiple ways to write γ as the catenation of two variable disjoint strongly-nested patterns γ' and γ'', we choose the one where γ' has minimal length.

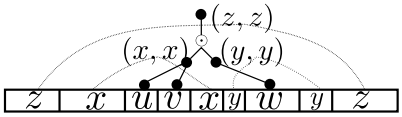


Figure 3 In the pattern, embedded in the z-blocks, two patterns are nested, which are again nested itself, namely the ones starting and ending in x and y respectively.

3. If $\gamma = \gamma'\gamma''\gamma'''$, where $\mathrm{var}(\gamma') = \mathrm{var}(\gamma''') = \{x\}$ and $x \notin \mathrm{var}(\gamma'')$ and γ', γ'', and γ''' start with a variable, then the tree T_γ consists of a node labelled with (γ', γ''') which has a single child: the tree $T_{\gamma''}$.

▶ **Lemma 21.** *The tree T_γ can be constructed in linear time $O(|\gamma|)$.*

Using the tree structure defined in the previous lemma, we can solve the matching problem for strongly-nested patterns efficiently.

▶ **Theorem 22.** MATCH *for* $\mathrm{PAT}_{\mathrm{nest}}$ *can be decided in $O(mn^3)$ time, where m is the length of the input pattern $\beta \in (X \cup \Sigma)^*$ and n is the length of the input word $w \in \Sigma^*$.*

In this algorithm, we assign the variables of β following, again, a local approach: we first assign all the variables in a subtree of T_β, and only then move on to its sibling (if it has one). We start with the trees consisting of single nodes, and then move on to more complex trees. Generally, we consider the trees in increasing order of their depth. The key observation is that the way the variables occurring in a subtree are assigned does not influence in any way the assignment of the variables outside this subtree (thus, enforcing a locality flavour) because a subtree only shares variables with its own subtrees.

Finally, the k-locality of strongly-nested patterns is considered. In particular, while it can be expected that the problem of computing the optimal k such that a pattern is k-local is intractable, it is shown that for strongly-nested patterns this can be done in polynomial time, and thus that they are not among the (expected) hard cases. Moreover, as a by-product of the algorithm, logarithmic bounds on the strict k-locality of strongly-nested patterns are given relative to their length, providing a clear formal comparison between the two classes.

The key to the algorithm is that, if we consider a factor $x..x$, it is assured that all other variables occurring in the factor do not occur outside as well, and thus the factor may be treated, to an extent, as independent from the rest of the pattern. Since such factors are fundamental to the remaining exposition, they are defined formally below.

▶ **Definition 23.** For a strongly-nested pattern $\alpha \in (X \cup \Sigma)^*$ and each $x \in \mathrm{var}(\alpha)$, let α_x be the shortest factor with $\alpha = \beta\alpha_x\gamma$ and $x \notin \mathrm{var}(\beta\gamma)$.

In other words, α_x is the factor of α from the leftmost to the rightmost occurrence of x. The approach is based around dynamic programming on the factors α_x, starting with the shortest (just blocks of the same variable). Precisely how this may be achieved is presented in Lemma 25 which gives the main recursive combinatorial insight. Firstly, however, it is necessary to distinguish between two types of marking sequences: those at which the edges are marked 'early' – i.e., as soon as the maximum number of marked blocks is reached – and those at which the edges are marked 'late'. Patterns permitting optimal marking sequences of the former variety are less likely to introduce a higher number of marked blocks, since at least one of the marked blocks may be absorbed by an existing marked block to the left/right.

▶ **Definition 24.** Let $\alpha \in (X \cup \Sigma)^+$ be a non-empty strictly k-local pattern. Let x and y be the leftmost and rightmost variables of α. Then α is *border priority markable* (BPM) if there

exists an optimal marking sequence for α (in the sense that no other marking sequence exists which requires strictly fewer marked blocks, or equivalently, that the maximum number of marked blocks required is k), such that whenever there are k distinct marked blocks, x and y are marked.

▶ **Lemma 25.** *Let $\alpha \in X^*$ be a (terminal-free) non-empty strongly-nested pattern and let $x \in \mathrm{var}(\alpha)$. Then either $\alpha_x \in \{x\}^+$, or there exist $y_1, y_2, \dots y_n \in \mathrm{var}(\alpha)$ such that $\alpha_x \in x^\mathfrak{b}\alpha_{y_1}\alpha_{y_2}\dots\alpha_{y_n}x^\mathfrak{b}$. Moreover, suppose that α_{y_i} is strictly k_i-local for $i \in [n]$ and let $\mu = \max_{i\in[n]}\{k_i\}$. Then:*

- *α_x is (strictly) μ-local if and only if there exists exactly one $i \in [n]$ such that $k_i = \mu$ with α_{y_i} is not BPM. Otherwise α_x is strictly $(\mu+1)$-local.*
- *α_x is BPM if and only if there exists exactly one $i \in [n]$ such that $k_i = \mu$, there exists at most one $j \in [n]$ such that $k_j = \mu - 1$ with α_{y_j} is not BPM, and α_{y_i} is BPM.*

▶ **Theorem 26.** *Given a strongly-nested pattern $\alpha \in (X \cup \Sigma)^*$, the smallest value k such that α is k-local can be computed in polynomial time.*

In addition to Theorem 26, it is also possible to infer the following bound on the strict k-locality of strongly-nested patterns from Lemma 25. Note the contrast to the general case for which a pattern of length n may be strictly $\frac{n}{2}$-local, as witnessed e.g., by $(xy)^{\frac{n}{2}}$.

▶ **Theorem 27.** *Let $\alpha \in (X \cup \Sigma)^*$ be a strongly-nested pattern, and let $k \in \mathbb{N}$ with $k > \log|\alpha|$. Then α is k-local. Moreover, for each $k \in \mathbb{N}$, there exist strongly-nested patterns of length $2^{k+1} + 2^{k-1} - 4$ which are strictly k-local.*

Finally, we propose the following extension of strongly-nested patterns. Let Π be a class of patterns for which we can decide in polynomial time both whether a pattern α is in Π and Match. We define *strongly-Π-nested patterns* as follows. Patterns $\beta \in \Pi$ are strongly-Π-nested patterns. If β_1, β_2 are variable-disjoint patterns, $\beta_1 \in \Pi$ and β_2 is a strongly-Π-nested pattern, $x \in X \setminus \mathrm{var}(\beta_1)$, $\gamma_1, \gamma_2 \in (\Sigma \cup \{x\})^*$, and $\beta_2'\beta_2'' = \beta_2$, then $\beta_2'\gamma_1\beta_1\gamma_2\beta_2''$ is a strongly-Π-nested pattern (i.e., we just shuffle $\gamma_1\beta_1\gamma_2$ inside β_2 to obtain $\beta_2'\gamma_1\beta_1\gamma_2\beta_2''$). Note that for $\gamma_1 = \gamma_2 = \beta_2' = \varepsilon$, we obtain that $\beta_1\beta_2$ is a strongly-Π-nested pattern.

It is not hard to see that one can decide whether a pattern is strongly-Π-nested in polynomial time. Essentially, to test whether β is such a pattern, we need to decide whether $\beta \in \Pi$ or there exist $1 \le i < j \le |\beta|$ such that $j - i + 1 < |\beta|$, $\beta[i..j]$ consists of a nested Π-pattern surrounded by two blocks of a variable x, and $\beta[1..i-1]\beta[j+1..|\beta|]$ is in Π. This can be clearly implemented in polynomial time. A similar strategy works for matching strongly-Π-nested patterns. Assume we want to match β to a word w, with $|\beta| = m$ and $|w| = n$. If $\beta \in \Pi$, then we just check if β matches w. Otherwise, for β there exist $1 \le i < j \le m$ such that $j - i + 1 < m$, $\beta[i..j]$ consists of a strongly-Π-nested pattern surrounded by two patterns containing only the variable x, and $\beta[1..i-1]\beta[j+1..n] \in \Pi$. Then we match first $\beta[i..j]$ to some factor $w[i'..j']$ of w, and then check if $\beta[1..i-1]\beta[j+1..m]$, which is in Π, can be matched to $w[1..i'-1]w[j'+1..n]$. Again, this clearly works in polynomial time.

6 Conclusions

We introduce the classes of k-local and strongly-nested patterns. We give polynomial time algorithms (assuming k is treated as constant) for the membership of these classes and for the matching problem, in both cases gaining a significant improvement compared to existing

algorithms for more general classes (i.e., those given in [25]). We have also considered the structure of patterns belonging to these classes, giving characterizations for patterns which are 1- and 2-local, as well as an optimized algorithm for matching 1-local patterns. We leave two interesting open problems outstanding, namely finding lower bounds for the time needed to decide whether a pattern is k-local, and for matching k-local patterns.

Acknowledgements. The authors wish to thank the referees of the paper for their helpful remarks and suggestions, in particular for their insightful comments on patterns with bounded treewidth which have helped to place the classes in the present paper more precisely within the literature.

References

1 Amihood Amir and Igor Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.

2 Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

3 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37, 2012.

4 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

5 Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Angelika Steger, and Thomas Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261:119–156, 2001.

6 Henning Fernau, Florin Manea, Robert Mercaş, and Markus L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. 32nd Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 302–315, 2015.

7 Henning Fernau, Florin Manea, Robert Mercaş, and Markus L. Schmid. Revisiting Shinohara's algorithm for computing descriptive patterns. *Theoretical Computer Science*, 2016. Accepted for publication. Preliminary version: `http://www.informatik.uni-trier.de/~schmid/preprints/journals/2017_TCS_preprint.pdf` .

8 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015.

9 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory of Computing Systems*, 2015.

10 Dominik D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems*, 53:159–193, 2013.

11 Dominik D. Freydenberger and Daniel Reidenbach. Inferring descriptive generalisations of formal languages. *Journal of Computer and System Sciences*, 79:622–639, 2013.

12 Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O'Reilly, Sebastopol, CA, third edition, 2006.

13 Juhani Karhumäki, Wojciech Plandowski, and Filippo Mignosi. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.

14 Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In *Proceedings of the 2nd Annual Conference on Learning Theory, COLT*, pages 57–71, 1989.

15 Satoshi Kobayashi and Takashi Yokomori. On approximately identifying concept classes in the limit. In *Proceedings of the 6th International Conference on Algorithmic Learning Theory, ALT*, volume 997 of *LNCS*, pages 298–312, 1995.

16 Dmitry Kosolobov, Florin Manea, and Dirk Nowotka. Detecting one-variable patterns. *CoRR*, abs/1604.00054, 2016. to appear in SPIRE 2017.

17 Stefan Lange and Rolf Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

18 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.

19 M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.

20 Alexandru Mateescu and Arto Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique Théoretique et Applications*, 28:233–253, 1994.

21 Zeinab Mazadi, Ziyuan Gao, and Sandra Zilles. Distinguishing pattern languages with membership examples. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA*, volume 8370 of *LNCS*, pages 528–540, 2014.

22 Yen K. Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.

23 Sebastian Ordyniak and Alexandru Popa. A parameterized study of maximum generalized pattern matching problems. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC*, 2014.

24 Daniel Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

25 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded treewidth. *Information and Computation*, 239:87–99, 2014.

26 Rüdiger Reischuk and Thomas Zeugmann. Learning one-variable pattern languages in linear average time. In *Proceedings of the 11th Annual Conference on Computational Learning Theory, COLT*, pages 198–208, 1998.

27 Markus L. Schmid. A note on the complexity of matching patterns with variables. *Information Processing Letters*, 113(19):729–733, 2013.

28 Takeshi Shinohara. Polynomial time inference of pattern languages and its application. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 191–209, 1982.

29 Takeshi Shinohara and Setsuo Arikawa. Pattern inference. In K.P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*, volume 961 of *LNAI*, pages 259–291, 1995.

30 Takeshi Shinohara and Hiroki Arimura. Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science*, 241:191–209, 2000.

On Symbolic Heaps Modulo Permission Theories

Stéphane Demri[1], Etienne Lozes[2], and Denis Lugiez[3]

1 **LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France**
demri@lsv.fr
2 **LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France**
lozes@lsv.fr
3 **LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay and Aix-Marseille Univ, LIF, CNRS, Marseille, France**
denis.lugiez@univ-amu.fr

Abstract

We address the entailment problem for separation logic with symbolic heaps admitting list predicates and permissions for memory cells that are essential to express ownership of a heap region. In the permission-free case, the entailment problem is known to be in P. Herein, we design new decision procedures for solving the satisfiability and entailment problems that are parameterised by the permission theories. This permits the use of solvers dealing with the permission theory at hand, independently of the shape analysis. We also show that the entailment problem without list predicates is coNP-complete for several permission models, such as counting permissions and binary tree shares but the problem is in P for fractional permissions. Furthermore, when list predicates are added, we prove that the entailment problem is coNP-complete when the entailment problem for permission formulae is in coNP, assuming the write permission can be split into as many read permissions as desired. Finally, we show that the entailment problem for any Boolean permission model with infinite width is coNP-complete.

1998 ACM Subject Classification D.2.4, Software/Program Verification, F.3. Logics and Meaning of Programs

Keywords and phrases separation logic, entailment, permission, reasoning modulo theories

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.25

1 Introduction

Separation logics with permissions. In program verification, proving properties of the memory is one of the most difficult tasks and separation logic has been devised for this goal [13]. Separation logic with permissions [4] can express that the ownership of a given heap region is shared with other threads. A permission can be thought of as a "quantity of ownership" associated to each cell of the heap. This quantity prescribes whether write accesses are allowed or not on this cell and how such a write access may be restored in the future. This abstract notion has lead to many permission theories and separation logics, including fractional permissions [5], token-based permissions [4], combinations of the two, binary tree shares [7], and yet some other models. Separation logic with permissions is supported by several tools like VeriFast [11], Hip/Sleek [10], or Heap-Hop [15]. Usually, these tools support only one permission model and demand that permissions are explicit values. For instance, in a tool that supports fractional permissions, to express that a cell x is shared by two threads for read access, one may write $x \overset{0.3}{\mapsto} y$ and $x \overset{0.7}{\mapsto} y$ making an arbitrary choice for permissions (0.3 and 0.7) when a better approach would use $x \overset{\alpha}{\mapsto} y$ and $x \overset{\beta}{\mapsto} y$ and the

constraint $\mathbf{1} = \alpha + \beta$. This hides the logical structure of the proof and ties it to a specific arbitrary permission model.

Our motivations. We wish to get rid of the use of explicit permission models and to provide a separation logic with permissions which can use symbolic permission expressions such as $\mathbf{1} = \alpha + \beta$. Furthermore, we aim at lifting the results obtained so far for separation logic with lists but without permissions to separation logic with lists and symbolic permissions.

Our contributions. We devise a separation logic based on symbolic heaps with list predicates [3] modulo an unspecified permission theory (containing separation logic without permission as an instance). As far as we know, a uniform treatment with both features is new. We give generic decision procedures modulo a permission theory $\mathfrak{P}$ for the satisfiability problem SATSH($\mathfrak{P}$), and for the entailment problem ENTSH($\mathfrak{P}$). Then we simply instantiate the permission theory by the desired theory (fractional models, token model, binary tree-share model,...). This approach has many advantages: (a) the reasoning on the spatial part is separated from the reasoning on permissions, (b) the latter part can be discharged to a dedicated solver, for instance any SMT specialised in the relevant permission theory (see e.g. [1]; and also [12] for the fractional case), and (c) we obtain optimal worst-case complexity results (obviously, the whole complexity depends on the complexity of the permission theory of interest). Since our logic contains the constant $\top$, we can treat both the intuitionistic and the non-intuitionistic case of the entailment problem in a uniform setting, see e.g. [6, 9]. Let us detail more precisely the technical contributions as well as the plan of the paper.

Outline of the paper. Permissions and separation logic with lists and permissions are introduced in Section 2. In Section 3, we treat separation logic with permissions but without lists and we give PTIME algorithms for SATSH($\mathfrak{P}$) and ENTSH($\mathfrak{P}$) using an oracle for the corresponding problems on permission theories. As a byproduct, SATSH($\mathfrak{P}_{\mathsf{Boy}}$) and ENTSH($\mathfrak{P}_{\mathsf{Boy}}$) without list predicates are in PTIME for the fractional model $\mathfrak{P}_{\mathsf{Boy}}$. In Section 4, we prove that SATSH($\mathfrak{P}$) is NP-hard and that ENTSH($\mathfrak{P}$) is coNP-hard even for a permission theory $\mathfrak{P}$ which is in PTIME, showing a complexity gap between the logic without permissions and the logic with permissions. At the end of Section 4, we design a non-deterministic polynomial-time procedure solving ENTSH($\mathfrak{P}$) (fully parametrised by the entailment problem for the permission theory $\mathfrak{P}$). A key ingredient is the notion of SL-graphs that are used to abstract formulae and several variants of homomorphisms between graphs used to prove the entailment property. This approach is clearly inspired by [6] but on one hand we can take advantage of nondeterminism since there is little hope for a PTIME algorithm, and on the other hand permissions lead to technical complications (such as the need to respect the linearisation induced by an SL-graph). In Section 5, we give our results on permission theories: (i) the fractional model $\mathfrak{P}_{\mathsf{Boy}}$ has PTIME satisfiability and entailment problems, (ii) we introduce the notion of Boolean permission models $\mathfrak{P}_{\mathbb{B}}$ that encompasses all classical permission models but the trivial one and $\mathfrak{P}_{\mathsf{Boy}}$ and (iii) we prove that SAT($\mathfrak{P}_{\mathbb{B}}$) is NP-complete and ENT($\mathfrak{P}_{\mathbb{B}}$) is coNP-complete in Boolean permission models $\mathfrak{P}_{\mathbb{B}}$ that have an infinite width (which is the case for the aforementionned models). Section 6 concludes the paper.

2 Preliminaries

We introduce permission formulae and permission models which are the building blocks for defining symbolic heaps with permissions and their related decision problems.

2.1 Permission models

Permission formulae are defined by the grammar below:

$$\begin{array}{lll} p ::= & \mathbf{1} \mid \alpha \mid p \oplus p & \text{(permission term)} \\ A ::= & \top \mid p = p \mid p \leq p \mid \mathsf{defined}(p) \mid A \wedge A & \text{(permission formula)} \end{array}$$

where $\mathtt{PVar} = \{\alpha, \beta, \dots\}$ is a countably infinite set of **permission variables**. Permission formulae are interpreted in permission models, defined below.

▶ **Definition 1.** A **permission model** is a tuple $\mathfrak{P} = (P_{\mathfrak{P}}, \mathbf{1}_{\mathfrak{P}}, \oplus_{\mathfrak{P}})$ such that

- $P_{\mathfrak{P}} = \{\pi, \dots\}$ is a set of **permissions**,
- $\mathbf{1}_{\mathfrak{P}} \in P_{\mathfrak{P}}$ is a distinguished permission called the **write** permission or the **total** permission,
- $\oplus_{\mathfrak{P}} : P_{\mathfrak{P}} \times P_{\mathfrak{P}} \to P_{\mathfrak{P}}$ is a partial composition that is cancellative, commutative and associative,[1]
- the relation $<_{\mathfrak{P}} \stackrel{\text{def}}{=} \{(\pi', \pi) \mid \pi = \pi' \oplus_{\mathfrak{P}} \pi'' \text{ for some } \pi''\}$ is irreflexive and transitive, with maximum element $\mathbf{1}_{\mathfrak{P}}$.

An example of permission model is Boyland's fractional model $\mathfrak{P}_{\mathsf{Boy}} = ((0,1], 1, \oplus_{\mathfrak{P}_{\mathsf{Boy}}})$ [5], where $\pi \oplus_{\mathfrak{P}_{\mathsf{Boy}}} \pi' \stackrel{\text{def}}{=} \pi + \pi'$ is defined when the sum is at most 1. The **width** of a permission model $\mathfrak{P}$ is $\mathsf{width}(\mathfrak{P}) \in \mathbb{N} \cup \{\omega\}$ such that $\mathsf{width}(\mathfrak{P}) \stackrel{\text{def}}{=} \sup\{n \geq 1 \mid \exists\, \pi_1, \dots, \pi_n \in P_{\mathfrak{P}} \text{ such that } \pi_1 \oplus \dots \oplus \pi_n = \mathbf{1}_{\mathfrak{P}}\}$.

Given $\mathfrak{P} = (P_{\mathfrak{P}}, \mathbf{1}_{\mathfrak{P}}, \oplus_{\mathfrak{P}})$, a **$\mathfrak{P}$-interpretation** is a map $\iota : \mathtt{PVar} \to P_{\mathfrak{P}}$. The map ι is extended to a partial map from the set of permission terms to $P_{\mathfrak{P}}$ so that $\iota(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1}_{\mathfrak{P}}$ and $\iota(p \oplus p') \stackrel{\text{def}}{=} \iota(p) \oplus_{\mathfrak{P}} \iota(p')$ if $\iota(p)$, $\iota(p')$ and $\iota(p) \oplus_{\mathfrak{P}} \iota(p')$ are defined. Otherwise $\iota(p \oplus p')$ is undefined. We may write $[\![p]\!]_\iota$ for $\iota(p)$ and $\iota \models A$ to denote that ι satisfies the permission formula A, following the clauses below:

- always $\iota \models \top$; $\iota \models \mathsf{defined}(p) \stackrel{\text{def}}{\Leftrightarrow} \iota(p)$ is defined; $\iota \models A \wedge A' \stackrel{\text{def}}{\Leftrightarrow} \iota \models A$ and $\iota \models A'$.
- $\iota \models p = p' \stackrel{\text{def}}{\Leftrightarrow}$ both $\iota(p)$ and $\iota(p')$ are defined and $\iota(p) = \iota(p')$.
- $\iota \models p \leq p' \stackrel{\text{def}}{\Leftrightarrow}$ both $\iota(p)$ and $\iota(p')$ are defined and, either $\iota(p) = \iota(p')$ or $\iota(p) <_{\mathfrak{P}} \iota(p')$.

For example, the permission formula $\alpha \oplus \alpha = \mathbf{1}$, is satisfied by the $\mathfrak{P}_{\mathsf{Boy}}$-interpretation ι defined by $\iota(\alpha) = 0.5$. We write $\bot$ to denote $\mathbf{1} \oplus \mathbf{1} = \mathbf{1}$. Observe that $\iota \not\models \bot$ for all ι (by irreflexivity of $<_{\mathfrak{P}}$).

2.2 Separation logic with permissions

A **symbolic heap** with list predicates and symbolic permissions is a formula (Π, Σ) where Π is a **pure formula** and Σ a **spatial formula** according to the grammar below:

$$\begin{array}{lll} \Pi ::= & \top \mid x = y \mid x \neq y \mid A \mid \Pi \wedge \Pi & \text{(pure formula)} \\ \Sigma ::= & \mathsf{emp} \mid \top \mid x \overset{p}{\mapsto} y \mid \mathsf{lseg}_p(x, y) \mid \Sigma * \Sigma & \text{(spatial formula)} \end{array}$$

[1] in particular, whenever a sum $\pi_1 \oplus_{\mathfrak{P}} \pi_2 \dots \oplus_{\mathfrak{P}} \pi_n$ is defined, each subsum $\pi_{i_1} \oplus_{\mathfrak{P}} \dots \oplus_{\mathfrak{P}} \pi_{i_k}$ for each $\{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$ is defined.

where $\mathtt{LVAR} = \{x, y, \dots\}$ is a countably infinite set of **location/program variables**. We write Π_{pe} and Π_{pv} to denote respectively the permission constraints and the program variable constraints that appear in Π, so that Π is logically equivalent to $\Pi_{pe} \wedge \Pi_{pv}$. We write $\mathtt{LVAR}(\varphi)$ [resp. $\mathtt{PVar}(\varphi)$] to denote the set of location [resp. permission] variables occurring in φ.

▶ **Example 2.** The following symbolic heaps are used throughout the paper.
$(\Pi_1, \Sigma_1) \stackrel{\text{def}}{=} (\bigwedge_{1 \le i < j \le 3} x_i \neq x_j, \mathsf{lseg}_\alpha(x_1, x_3))$
$(\Pi_2, \Sigma_2) \stackrel{\text{def}}{=} (\bigwedge_{1 \le i < j \le 3} x_i \neq x_j, \mathsf{lseg}_\alpha(x_1, x_2) * \mathsf{lseg}_\alpha(x_2, x_3))$
$(\Pi_3, \Sigma_3) \stackrel{\text{def}}{=} (\bigwedge_{1 \le i < j \le 3} x_i \neq x_j, \mathsf{lseg}_{\alpha\oplus\alpha}(x_1, x_3) * \mathsf{lseg}_\alpha(x_3, x_2) * \mathsf{lseg}_\alpha(x_2, x_1))$.

Let $\mathfrak{P} = (P_\mathfrak{P}, \mathbf{1}_\mathfrak{P}, \oplus_\mathfrak{P})$ be a fixed permission model and let $\mathsf{Loc} = \{\ell, \dots\}$ be a countably infinite set of locations (by default, $\mathsf{Loc} = \mathbb{N}$). A **$\mathfrak{P}$-memory state** is a triple (s, h, ι) where:

- s is a store i.e. a function $s : \mathtt{LVAR} \to \mathsf{Loc}$ that assigns to each variable a location,
- h is a **$\mathfrak{P}$-heap** i.e. a partial function with a finite domain $h : \mathsf{Loc} \rightharpoonup_{\mathsf{fin}} P_\mathfrak{P} \times \mathsf{Loc}$,
- ι is a $\mathfrak{P}$-interpretation.

Intuitively, $h(\ell) = (\pi, \ell')$ holds if the cell at address ℓ is allocated and points to the location ℓ', and that the thread that owns ℓ has permission π on the cell ℓ.

Before defining the semantics of symbolic heaps, we define the composition of $\mathfrak{P}$-heaps. The **composition** $h_1 \bullet h_2$ of two $\mathfrak{P}$-heaps h_1 and h_2 is defined whenever there is no $\ell \in \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$ with $h_1(\ell) = (\pi_1, \ell_1)$ and $h_2(\ell) = (\pi_2, \ell_2)$ such that either $\ell_1 \neq \ell_2$ or $\pi_1 \oplus_\mathfrak{P} \pi_2$ is undefined. When $h_1 \bullet h_2$ is defined, say equal to the $\mathfrak{P}$-heap h, it takes the unique value satisfying the conditions below:

- if $\ell \notin \mathrm{dom}(h_1) \cup \mathrm{dom}(h_2)$, then $\ell \notin \mathrm{dom}(h)$,
- if $\ell \in \mathrm{dom}(h_i) \setminus \mathrm{dom}(h_j)$, then $\ell \in \mathrm{dom}(h)$ and $h(\ell) = h_i(\ell)$ (for all $i \neq j \in \{1, 2\}$),
- if $\ell \in \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$, then $\ell \in \mathrm{dom}(h)$ and $h(\ell) = (\pi_1 \oplus_\mathfrak{P} \pi_2, \ell')$ with $h_1(\ell) = (\pi_1, \ell')$, $h_2(\ell) = (\pi_2, \ell')$ and $\pi_1 \oplus_\mathfrak{P} \pi_2$ is defined

The composition of heaps is partial, commutative, associative, and cancellative. We write $h' \sqsubseteq h$ if there is h'' so that $h = h' \bullet h''$ and we also write $h' \sqsubset h$ whenever $h' \sqsubseteq h$ and $h' \neq h$. The satisfaction relations $s, h, \iota \models_\mathfrak{P} \Sigma$ or $s, h, \iota \models_\mathfrak{P} \Pi$ are defined below:

$s, h, \iota \models_\mathfrak{P} \top$	always	
$s, h, \iota \models_\mathfrak{P} x = y$	iff	$s(x) = s(y)$
$s, h, \iota \models_\mathfrak{P} x \neq y$	iff	$s(x) \neq s(y)$
$s, h, \iota \models_\mathfrak{P} A$	iff	$\iota \models A$
$s, h, \iota \models_\mathfrak{P} \Pi_1 \wedge \Pi_2$	iff	$s, h, \iota \models_\mathfrak{P} \Pi_1$ and $s, h, \iota \models_\mathfrak{P} \Pi_2$
$s, h, \iota \models_\mathfrak{P} \mathtt{emp}$	iff	$\mathrm{dom}(h) = \emptyset$
$s, h, \iota \models_\mathfrak{P} x \overset{p}{\mapsto} y$	iff	$\mathrm{dom}(h) = \{s(x)\}$, $[\![p]\!]_\iota$ is defined, and $h(s(x)) = ([\![p]\!]_\iota, s(y))$
$s, h, \iota \models_\mathfrak{P} \mathsf{lseg}_p(x, y)$	iff	$[\![p]\!]_\iota$ is defined, and either $(s(x) = s(y)$ and $\mathrm{dom}(h) = \emptyset)$
		or $h = \{\ell_0 \mapsto ([\![p]\!]_\iota, \ell_1), \ell_1 \mapsto ([\![p]\!]_\iota, \ell_2), \dots, \ell_{n-1} \mapsto ([\![p]\!]_\iota, \ell_n)\}$
		with $n \geq 1$, $\ell_0 = s(x)$, $\ell_n = s(y)$ and for all $i \neq j \in [0, n]$, $\ell_i \neq \ell_j$
$s, h, \iota \models_\mathfrak{P} \Sigma_1 * \Sigma_2$	iff	there are subheaps h_1, h_2 such that
		$h = h_1 \bullet h_2$, $s, h_1, \iota \models_\mathfrak{P} \Sigma_1$, and $s, h_2, \iota \models_\mathfrak{P} \Sigma_2$.

Our definitions of symbolic heaps and models encompass the standard definitions without permissions: choose $\mathfrak{P}_\mathbf{1} = (\{\mathbf{1}\}, \mathbf{1}, \oplus_{\mathfrak{P}_\mathbf{1}})$ that has only the write permission and the always undefined composition. By way of example, given the permission model $\mathfrak{P}_{\mathsf{Boy}}$, let (s, h, ι) be defined by $s(x_1) = 1, s(x_2) = 2, s(x_3) = 3$, $h = \{1 \mapsto (0.5, 3), 3 \mapsto (0.25, 2), 2 \mapsto (0.25, 1)\}$ and $\iota(\alpha) = 0.25$. Then, taking the (Π_i, Σ_i)'s from Example 2, we have $s, h, \iota \not\models_{\mathfrak{P}_{\mathsf{Boy}}} (\Pi_1, \Sigma_1)$ but $s, h, \iota \models_{\mathfrak{P}_{\mathsf{Boy}}} (\Pi_2, \Sigma_2)$ (since 0.5 can be split into $0.25 + 0.25$) and $s, h, \iota \models_{\mathfrak{P}_{\mathsf{Boy}}} (\Pi_3, \Sigma_3)$.

$$
\begin{array}{lllll}
(\textsc{Subst}) & (\Pi, \Sigma) & \Longrightarrow & (\Pi, \Sigma[y/x]) & \text{if } \Pi \models x = y, \{x, y\} \subseteq \mathtt{LVAR}(\Sigma) \\
(\textsc{Merge}) & (\Pi, \Sigma * x \overset{p}{\mapsto} y * x \overset{p'}{\mapsto} z) & \Longrightarrow & (\Pi \wedge y = z, \Sigma * x \overset{p \oplus p'}{\mapsto} y) & \\
(\textsc{Fail}) & (\Pi, \Sigma) & \Longrightarrow & \bot \quad \text{if } \Pi_{pv} \models x \neq y \text{ and } \Pi_{pv} \models x = y & \\
(\textsc{Empty}) & (\Pi, \Sigma * \mathtt{emp}) & \Longrightarrow & (\Pi, \Sigma) \quad \text{if non-empty } \Sigma & \\
(\textsc{True}) & (\Pi, \Sigma * \top * \top) & \Longrightarrow & (\Pi, \Sigma * \top) & \\
(\textsc{Mergelist}) & (\Pi, \Sigma * \mathsf{lseg}_p(x, y) * \mathsf{lseg}_{p'}(x, y) & \Longrightarrow & (\Pi, \Sigma * \mathsf{lseg}_{p \oplus p'}(x, y)) &
\end{array}
$$

Figure 1 Rewrite system R.

Satisfiability and entailment problems. A symbolic heap (Π, Σ) is **$\mathfrak{P}$-satisfiable** if there is a $\mathfrak{P}$-memory state (s, h, ι) such that $s, h, \iota \models_{\mathfrak{P}} \Pi$ and $s, h, \iota \models_{\mathfrak{P}} \Sigma$ and we say that (s, h, ι) is a **$\mathfrak{P}$-model** of (Π, Σ). Two symbolic heaps (Π, Σ) and (Π', Σ') are equivalent, written $(\Pi, \Sigma) \equiv_{\mathfrak{P}} (\Pi', \Sigma')$, if they have the same $\mathfrak{P}$-models. The **satisfiability problem w.r.t. $\mathfrak{P}$**, written SATSH($\mathfrak{P}$), takes as input (Π, Σ) and asks whether (Π, Σ) has a $\mathfrak{P}$-model. The **entailment problem w.r.t. $\mathfrak{P}$** written ENTSH($\mathfrak{P}$)) takes as input two symbolic heaps (Π, Σ) and (Π', Σ') and asks whether every $\mathfrak{P}$-model of (Π, Σ) is a $\mathfrak{P}$-model of (Π', Σ') (written $(\Pi, \Sigma) \models_{\mathfrak{P}} (\Pi', \Sigma')$). In the paper, the decision procedures are parameterised by the corresponding problems in the permission model $\mathfrak{P}$, written SAT($\mathfrak{P}$) and ENT($\mathfrak{P}$).

3 Reasoning Modulo Permission Theories Without List Predicates

3.1 Normalising formulae

In Figure 1, we present a set R of rewrite rules that are used to normalise formulae. The reduction $\Longrightarrow$ is the rewrite relation associated to R and $\Longrightarrow^*$ is its reflexive and transitive closure. Note that if a rewrite sequence starts from a symbolic heap not containing an expression $\mathsf{lseg}_p(x, y)$, then the rule Mergelist never applies. We write $|(\Pi, \Sigma)|$ to denote the **size** of the symbolic heap for some reasonably succinct encoding.

▶ **Lemma 3.** *The rewrite relation $\Longrightarrow$ has the following properties.*

- *If $(\Pi, \Sigma) \Longrightarrow (\Pi', \Sigma')$ then $(\Pi, \Sigma) \equiv (\Pi', \Sigma')$.*
- *Any rewrite sequence starting from (Π, Σ) terminates in time $\mathcal{O}(|(\Pi, \Sigma)|)$.*

The proof of the first part is a direct analysis of the rules according to the semantics of symbolic heaps and the termination proof is straightforward. Note that the rule Subst cannot be applied indefinitely because of the second side-condition. From now on, unless otherwise stated, **normal form** refers to a normal form with respect to $\Longrightarrow$.

3.2 Satisfiability and entailment for symbolic heaps without lists

We give our first results for symbolic heaps with permission but without lists. In the rest of this section, we consider symbolic heaps without lists. Given a spatial formula Σ, we denote by $\mathsf{defined}(\Sigma)$ the conjunction of formulae $\mathsf{defined}(p)$ for all p occuring in Σ.

▶ **Lemma 4.** *Given (Π, Σ) in normal form, Π, Σ is satisfiable iff $(\Pi_{pe} \wedge \mathsf{defined}(\Sigma))$ is satisfiable.*

Consequently, we can provide complexity upper bounds for SATSH($\mathfrak{P}$).

▶ **Theorem 5.** *Let $\mathfrak{P}$ be a permission model and $\mathcal{C} \supseteq$ PTime be a complexity class such that* SAT($\mathfrak{P}$) *is in $\mathcal{C}$. Then* SATSH($\mathfrak{P}$) *restricted to symbolic heaps without list predicates is in $\mathcal{C}$.*

$$\text{(Align)} \quad (\Pi, \Sigma_l * x \overset{p}{\mapsto} y, \Sigma_r * x' \overset{p'}{\mapsto} y') \implies (\Pi \wedge \mathsf{defined}(p), \Sigma_l, \Sigma_r) \quad \text{if } \Pi \models p = p' \wedge x = x' \wedge y = y'$$

$$\text{(Substract)} \quad \left(\Pi\ ,\ \Sigma_l * x \overset{p}{\mapsto} y\ ,\ \Sigma_r * x' \overset{p'}{\mapsto} y'\right) \implies \left(\Pi \wedge p = p' \oplus \alpha\ ,\ \Sigma_l * x \overset{\alpha}{\mapsto} y\ ,\ \Sigma_r\right)$$
$$\text{if } \Pi \models x = x' \wedge y = y',\ \Pi \wedge \mathsf{defined}(\Sigma_l) \wedge p = p' \oplus \alpha \text{ is satisfiable}, \alpha \notin \mathtt{PVar}(\Pi, \Sigma_l, \Sigma_r)$$

Figure 2 Rewrite rules for triples (Π, Σ, Σ')

Let us now address the entailment problem $(\Pi_l, \Sigma_l) \models (\Pi_r, \Sigma_r)$. First, we restrict our attention to instances of the form $(\Pi_l, \Sigma_l) \models (\top, \Sigma_r)$, where (Π_l, Σ_l) *is in normal form*. An entailment $(\Pi_l, \Sigma_l) \models (\top, \Sigma_r)$ holds if there is a map from the points-to predicates $x' \overset{p'}{\mapsto} y'$ that occur in Σ_r to the $x \overset{p}{\mapsto} y$ that occur in Σ_l, such that the sum of all permissions terms p' mapped to a given $x \overset{p}{\mapsto} y$ is smaller or equal to p, with an equality required if $\top$ does not occur in Σ_r. We represent $(\Pi_l, \Sigma_l) \models (\top, \Sigma_r)$ by a triple $(\Pi_l, \Sigma_l, \Sigma_r)$, and we check the existence of such a map by means of the rewrite rules Align and Substract of Figure 2. Intuitively, we remove each points-to predicate of Σ_r, one by one, until Σ_r is trivial. This new rewrite relation, denoted by $\Longrightarrow_{AS}$, terminates in at most $|\Sigma_r|$ steps, and it preserves the entailment validity : if $(\Pi_l, \Sigma_l, \Sigma_r) \Longrightarrow_{AS} (\Pi'_l, \Sigma'_l, \Sigma'_r)$ then $(\Pi_l, \Sigma_l) \models (\top, \Sigma_r)$ iff $(\Pi'_l, \Sigma'_l) \models (\top, \Sigma'_r)$.

In order to check an entailment $(\Pi_l, \Sigma_l) \models (\Pi_r, \Sigma_r)$ where Π_r is not necessarily $\top$, we check on the one hand whether $(\Pi_l, \Sigma_l) \models \Pi_r$, and on the other hand, using rules Align and Substract, whether $(\Pi_l, \Sigma_l) \models \Sigma_r$, which is implemented by the algorithm below.

▶ **Lemma 6.** *The algorithm terminates, and returns* **true** *iff* $(\Pi, \Sigma) \models (\Pi', \Sigma')$.

▶ **Theorem 7.** *The problem* ENTSH($\mathfrak{P}$) *restricted to symbolic heaps without list predicates can be decided in polynomial time with an oracle for* ENT($\mathfrak{P}$).

Algorithm 1: Entailment checking without lists

input two symbolic heaps (Π, Σ) and (Π', Σ') without list predicates
output returns **true** if $(\Pi, \Sigma) \models (\Pi', \Sigma')$, and returns **false** otherwise
  put (Π, Σ) in normal form
  if $(\Pi, \Sigma) = \bot$ or $\Pi_{pe} \wedge \mathsf{defined}(\Sigma)$ is not $\mathfrak{P}$-satisfiable **then return true**
  if $\Pi_{pe} \wedge \mathsf{defined}(\Sigma) \not\models \Pi'_{pe}$ **then return false**
  for all atomic formulae φ of Π'_{pv} **do**
    let (Π'', Σ'') be the normal form of $(\Pi \wedge \neg\varphi, \Sigma)$
    if $(\Pi'', \Sigma'') \neq \bot$ and $\Pi''_{pe} \wedge \mathsf{defined}(\Sigma'')$ is $\mathfrak{P}$-satisfiable **then return false**
  end for
  put $\left(\Pi, \Sigma, \Sigma'\right)$ in normal form w.r.t. $\Longrightarrow_{AS}$
  put (Π, Σ') in normal form w.r.t. $\Longrightarrow$
  return $\left(\Sigma' = \top\right)$ or $\left(\Sigma = \Sigma' = \mathsf{emp}\right)$

Using the results from Section 5, it follows that ENTSH($\mathfrak{P}_{\mathsf{Boy}}$) restricted to symbolic heaps without list predicates is in PTime.

4 Reasoning on Symbolic Heaps with Lists and Permissions

Below, we design algorithms for SATSH($\mathfrak{P}$) and ENTSH($\mathfrak{P}$) respectively, parameterised by decision problems for $\mathfrak{P}$. Assuming that Σ and Σ' are $\top$-free and $\mathtt{emp}$-free spatial formulae, we introduce the following subproblems of ENTSH($\mathfrak{P}$):

- $(\Pi, \Sigma) \models_I (\Pi', \Sigma') \stackrel{\text{def}}{\Leftrightarrow} (\Pi, \Sigma * \top) \models (\Pi', \Sigma' * \top)$. (intuitionistic entailment)
- $(\Pi, \Sigma) \models_{NI} (\Pi', \Sigma') \stackrel{\text{def}}{\Leftrightarrow} (\Pi, \Sigma) \models (\Pi', \Sigma')$. (non-intuitionistic entailment)

Below, we provide the developments for $\models_{NI}$ only but all our results can be adapted for the full problems SATSH($\mathfrak{P}$) and ENTSH($\mathfrak{P}$). In the permission model $\mathfrak{P}_1$, SATSH($\mathfrak{P}_1$) and ENTSH($\mathfrak{P}_1$) are in PTIME [6] but untractability of SATSH($\mathfrak{P}$) and ENTSH($\mathfrak{P}$) happens quite quickly, even with the rather simple permission model $\mathfrak{P}_{\mathsf{Boy}}$. A positive consequence of Theorem 9 is that we can use non-determinism to get optimal complexity bounds.

4.1 Lower bounds

In this short section, we explain how the combination of list predicates and permission leads to NP/coNP-hardness. Let $G = (V, E)$ be an instance of the three-colorability problem, known to be NP-complete. W.l.o.g., we can assume that $V = \{x_1, \ldots, x_n\}$ for some $n \geq 1$ and E is a set of edges of the form $\{x_i, x_j\}$ with $i \neq j$. The hardness proof is by reduction from the three-colorability problem following a similar treatment when conjunctions are added to spatial formulae in the standard symbolic heap fragment, see [6, Section 5]. The main difference below rests on the replacement of Boolean conjunctions by separating conjunctions. Let Π_G be the pure formula $(\bigwedge_{\{x_i,x_j\}\in E}(x_i \neq x_j)) \wedge y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3$. Let Σ_G be the spatial formula $y_1 \stackrel{\alpha_0}{\mapsto} y_2 * y_2 \stackrel{\alpha'_0}{\mapsto} y_3 * y_3 \stackrel{\mathbf{1}}{\mapsto} y_1 \circledast_{x_i \in V} (\mathsf{lseg}_{\alpha_i}(y_1, x_i) * \mathsf{lseg}_{\alpha'_i}(x_i, y_3))$, where the α_i's and α'_i's are distinct permission variables.

▶ **Lemma 8.** *Assuming that* $\mathsf{width}(\mathfrak{P}) = \omega$, *$G$ has a three-coloring iff (Π_G, Σ_G) is satisfiable.*

Consequently, we get the following hardness results.

▶ **Theorem 9.** *If* $\mathsf{width}(\mathfrak{P}) = \omega$ *then* SATSH($\mathfrak{P}$) *is* NP*-hard and* ENTSH($\mathfrak{P}$) *is* coNP*-hard.*

Observe that (Π, Σ) is not $\mathfrak{P}$-satisfiable iff $(\Pi, \Sigma) \models_{\mathfrak{P}} x \neq x$. So, the NP-hardness of SATSH($\mathfrak{P}$) entails the coNP-hardness of ENTSH($\mathfrak{P}$).

4.2 SL-graphs and homomorphisms

We assume a fixed permission model $\mathfrak{P}$ and a fixed set of variables $\{x_1, \ldots, x_q\}$ with its ordering $x_1 < \ldots < x_q$. All the symbolic heaps are assumed to be built from $\{x_1, \ldots, x_q\}$. Given $\emptyset \neq X \subseteq \{x_1, \ldots, x_q\}$, $\min(X)$ denotes the variable in X with the minimal index.

Below, we introduce a notion of SL-graph that can be understood as a graphical representation of a symbolic heap in which the permission part of the pure formula is encoded directly by a permission formula, the location variable part of a pure formula is encoded by an inequality relation ($\stackrel{\neq}{\longleftrightarrow}$) and by a labelling ($L$). Besides, the atomic spatial formulae are encoded by the two relations $\rightarrow$ and $\Rightarrow$. Such structures are quite convenient to characterise entailment between symbolic heaps via homomorphisms, as it is done in the permission-free setting in [6]. Contrary to the developments in [6] that aim to reach a PTIME upper bound, nondeterminism will be essential below since this is the best we can hope for, see Theorem 9.

An **SL-graph** G is either $\bot$ or a tuple $(A, V, \rightarrow, \Rightarrow, \stackrel{\neq}{\longleftrightarrow}, L)$ such that

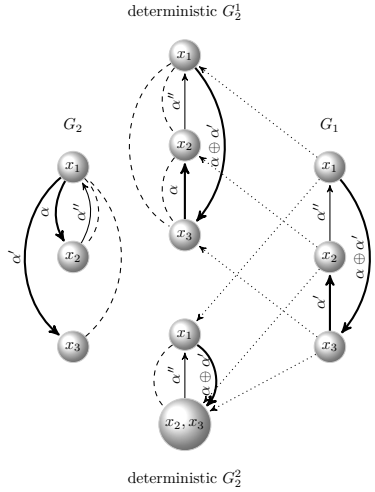


Figure 3 SL-graphs.

- A is a permission formula and V is a non-empty finite subset of $\mathbb{N}$ (the nodes).
- $\rightarrow$ and $\Rightarrow$ are finite subsets of $V \times PT \times V$ where PT is the set of permission terms. We also write $v \xrightarrow{p} v'$ [resp. $v \overset{p}{\Rightarrow} v'$] instead $(v, p, v') \in \rightarrow$ [resp. $(v, p, v') \in \Rightarrow$]. We require a functionality condition: $v \xrightarrow{p} v'$ implies the unicity of p and v'.
- $\overset{\neq}{\longleftrightarrow}$ is an irreflexive and symmetric binary relation on V.
- $L : \{x_1, \ldots, x_q\} \rightarrow V$ is a surjective labelling.

Given $v \in V$, we write $vars(v)$ to denote the (non-empty) set $\{x \mid L(x) = v\}$ and $var(v) \overset{\text{def}}{=} min(vars(v))$. As expected, the arrow $v \xrightarrow{p} v'$ [resp. $v \overset{p}{\Rightarrow} v'$] is intended to represent the atomic formula $x \overset{p}{\mapsto} x'$ [resp. $\mathsf{lseg}_p(x, x')$] whenever $L(x) = v$ and $L(x') = v'$. Moreover, we write $v \overset{p}{\rightsquigarrow} v'$ whenever $v \xrightarrow{p} v'$ or $v \overset{p}{\Rightarrow} v'$. Our notion of SL-graph shares in spirit the one from [6] but there are essential differences (permission formulae and terms as well as slight simplifications). Figure 3 presents SL-graphs where dashed lines encode the inequality relation, thick arrows encode $\Rightarrow$, normal arrows encode $\rightarrow$ and the permission formulae are omitted.

Below, we provide a semantics to the SL-graphs by defining a symbolic heap for each SL-graph. Given $G = (A, V, \rightarrow, \Rightarrow, \overset{\neq}{\longleftrightarrow}, L)$, $(\mathsf{pure}(G), \mathsf{spatial}(G))$ denotes the symbolic heap defined from G:

$$\mathsf{pure}(G) \overset{\text{def}}{=} A \wedge \Big(\bigwedge_{x_i, x_j, L(x_i)=L(x_j)} x_i = x_j\Big) \wedge \Big(\bigwedge_{v \overset{\neq}{\longleftrightarrow} v'} var(v) \neq var(v')\Big).$$

$$\mathsf{spatial}(G) \overset{\text{def}}{=} \Big(\mathop{\Large *}_{v \xrightarrow{p} v'} var(v) \overset{p}{\mapsto} var(v')\Big) * \Big(\mathop{\Large *}_{v \overset{p}{\Rightarrow} v'} \mathsf{lseg}_p(var(v), var(v'))\Big).$$

An empty separating conjunction is understood as emp. If $G = \bot$, then the corresponding symbolic heap is $(x \neq x, \top)$. In the sequel, $\mathsf{spatial}(G)$ is also written $\mathop{\Large *}_{v \overset{p}{\rightsquigarrow} v' \in G} \chi(v \overset{p}{\rightsquigarrow} v')$, where $\chi(v \xrightarrow{p} v') \overset{\text{def}}{=} var(v) \overset{p}{\mapsto} var(v')$ and $\chi(v \overset{p}{\Rightarrow} v') \overset{\text{def}}{=} \mathsf{lseg}_p(var(v), var(v'))$.

An SL-graph $(A, V, \rightarrow, \Rightarrow, \overset{\neq}{\longleftrightarrow}, L)$ is **deterministic** iff for all $v \in V$, $\text{card}(\{v' \mid v \overset{p}{\rightsquigarrow} v'\}) \leq 1$ and, for all $v \neq v' \in V$, we have $v \overset{\neq}{\longleftrightarrow} v'$ (G_2^1 and G_2^2 are deterministic in Figure 3 unlike G_1 and G_2). A deterministic SL-graph can be viewed as a syntactic structure whose

interpretation stands between the $\mathfrak{P}$-memory states (thanks to the determinism and the syntactic nature of the inequality relation) and the SL-graphs (no $\mathfrak{P}$-interpretation and no permission values are involved). Each deterministic SL-graph carries a lot of structural properties about the $\mathfrak{P}$-memory states that satisfy it, which explains why this is a crucial structure to consider (see also the dependency graphs in [8]).

Let $G_1 = (A_1, V_1, \to_1, \Rightarrow_1, \overset{\neq}{\leftrightarrow}_1, L_1)$ and $G_2 = (A_2, V_2, \to_2, \Rightarrow_2, \overset{\neq}{\leftrightarrow}_2, L_2)$ be two SL-graphs with G_2 *being deterministic*. We introduce below a notion of precise homomorphism that will admit a counterpart in terms of entailment, see e.g. Theorem 11. A map $\mathfrak{f} : V_1 \to V_2$ is a **precise homomorphism** from G_1 to G_2 whenever the conditions below are satisfied:

(H0) $\mathfrak{f}$ is surjective.

(H1) $A^\star \models A_1$ where $A^\star = A_2 \wedge \bigwedge_i \mathsf{defined}(p_i) \wedge \bigwedge_j \mathsf{defined}(p'_j)$, $\to_2 = \{v_1 \overset{p_1}{\to}_2 v'_1, \ldots, v_n \overset{p_n}{\to}_2 v'_n\}$ and $\Rightarrow_2 = \{\{u_1 \overset{p'_1}{\Rightarrow}_2 u'_1, \ldots, u_m \overset{p'_m}{\Rightarrow}_2 u'_m\}\}$ (multiset notation).

(H2) For all $v \in V_1$, we have $vars(v) \subseteq vars(\mathfrak{f}(v))$.

(H3) For all $v, v' \in V_1$, $v \overset{\neq}{\leftrightarrow}_1 v'$ implies $\mathfrak{f}(v) \overset{\neq}{\leftrightarrow}_2 \mathfrak{f}(v')$.

(H4) For all $v, v' \in V_1$, $v \overset{p}{\to}_1 v'$ implies there is some permission term p' such that $\mathfrak{f}(v) \overset{p'}{\to}_2 \mathfrak{f}(v')$. We say that the edge $\mathfrak{f}(v) \overset{p'}{\to}_2 \mathfrak{f}(v')$ **contributes** to the edge $v \overset{p}{\to}_1 v'$.

(H5) For all $v, v' \in V_1$, $v \overset{p}{\Rightarrow}_1 v'$ implies either ($\mathfrak{f}(v) = \mathfrak{f}(v')$ and $A^\star \models \mathsf{defined}(p)$) or there is a path $v_0 \overset{p_0}{\rightsquigarrow}_2 v_1 \overset{p_1}{\rightsquigarrow}_2 v_2 \cdots \overset{p_{n-1}}{\rightsquigarrow}_2 v_n$ with $n \geq 1$, $v_0 = \mathfrak{f}(v)$, $v_n = \mathfrak{f}(v')$ and for all $i < j \in [0, n]$, $v_i \neq v_j$ (equivalent to $v_i \overset{\neq}{\leftrightarrow}_2 v_j$ since G_2 is deterministic). For each edge $v_k \overset{p_k}{\rightsquigarrow}_2 v_{k+1}$, we say that it **contributes** to the edge $v \overset{p}{\Rightarrow}_1 v'$. Since G_2 is deterministic, there is a unique path satisfying the above condition (if any).

(H6′) Each edge $v \overset{p'}{\rightsquigarrow}_2 v'$ in G_2 contributes to at least one edge of G_1, and we have $A^\star \models \oplus\{\{p \mid v \overset{p'}{\rightsquigarrow}_2 v' \text{ contributes to } u \overset{p}{\rightsquigarrow}_1 u' \in G_1\}\} = p'$.

Above, given a non-empty and finite multiset $T = \{\{p_1, \ldots, p_k\}\}$ of permission terms, we write $\oplus T$ instead of $p_1 \oplus \cdots \oplus p_k$ (the ordering of the terms is irrelevant because $\oplus$ is AC). Precise homomorphisms could be defined between two arbitrary SL-graphs (as done in [6]) but the unicity of the path in the condition (H5) is not anymore guaranteed. We assume that G_2 is deterministic to have unicity, which also leads to the right upper bounds for the complexity. The existence of a precise homomorphism $\mathfrak{f}$ implies that for all $u \overset{p}{\rightsquigarrow}_1 u' \in G_1$, we have $A^\star \models \mathsf{defined}(p)$, which is partly justified by $\mathsf{defined}(p_1 \oplus p_2) \models_{\mathfrak{P}} \mathsf{defined}(p_1) \wedge \mathsf{defined}(p_2)$. The dotted arrows in Figure 3 partly materialize two precise homomorphisms from G_1 to G_2^1 or to G_2^2 (assuming the permission formulae match).

A precise homomorphism $\mathfrak{f}$ is **strongly precise** $\overset{\text{def}}{\Leftrightarrow}$

(H1′) $A_2 = A_1 \wedge \bigwedge_{v \overset{p}{\rightsquigarrow} v' \text{in } G_1} \mathsf{defined}(p)$ (which implies (H1)),

(H5′) is equal to (H5) except that in the case $\mathfrak{f}(v) = \mathfrak{f}(v')$, we do not require that $A^\star \models \mathsf{defined}(p)$,

(H6″) each edge $v \overset{p'}{\rightsquigarrow}_2 v'$ in G_2 contributes to at least one edge of G_1, and we have $\oplus\{\{p \mid v \overset{p'}{\rightsquigarrow}_2 v' \text{ contributes to } u \overset{p}{\rightsquigarrow}_1 u' \in G_1\}\}$ equal to p' modulo AC (implying (H6′)).

In Figure 3, there is a strongly precise homomorphism from G_2 to G_2^1 [resp. to G_2^2] with the adequate permission formulae. Notably, checking whether $\mathfrak{f} : V_1 \to V_2$ is a [resp. strongly] precise homomorphism can be checked in coNP [resp. PTIME] when ENT($\mathfrak{P}$) is in coNP, which is useful to establish the complexity upper bounds.

4.3 From symbolic heaps to SL-graphs

Let (Π, Σ) be a symbolic heap, possibly with list predicates and, Σ is $\mathtt{emp}$-free and $\top$-free. Let us define the SL-graph $\mathsf{slg}(\Pi, \Sigma)$ as follows. If $(\Pi, \Sigma) \Longrightarrow^* \bot$, then $\mathsf{slg}(\Pi, \Sigma)$ is defined as the inconsistent SL-graph $\bot$ (see Section 3 for the definition of $\Longrightarrow$).

Otherwise, assume that $(\Pi, \Sigma) \Longrightarrow^* (\Pi', \Sigma')$ and (Π', Σ') is in normal form (i.e., no rule can be further fired from (Π', Σ')).

Let us define $\mathsf{slg}(\Pi, \Sigma) \stackrel{\text{def}}{=} (A, V, \rightarrow, \Rightarrow, \overset{\neq}{\leftrightarrow}, L)$ as follows:

- $A \stackrel{\text{def}}{=} \Pi'_{pe}$, $V \stackrel{\text{def}}{=} \{i \in [1, q] \mid \text{there is no } j < i \text{ such that } \Pi' \models x_i = x_j\}$.
- $L(x_i) \stackrel{\text{def}}{=} \min\{j \in V \mid \Pi' \models x_i = x_j\}$.
- If $x \stackrel{p}{\mapsto} y$ occurs in Σ', then $L(x) \stackrel{p}{\rightarrow} L(y)$; if $\mathsf{lseg}_p(x, y)$ occurs in Σ', then $L(x) \stackrel{p}{\Rightarrow} L(y)$.
- For all $i \neq j \in [1, q]$, we have $\{L(x_i), L(x_j)\} \in \overset{\neq}{\leftrightarrow} \stackrel{\text{def}}{\Leftrightarrow} \Pi' \models x_i \neq x_j$.

In Figure 3, $G_1 = \mathtt{slg}(\top, \mathsf{lseg}_{\alpha \oplus \alpha'}(x_1, x_3) * \mathsf{lseg}_{\alpha'}(x_3, x_2) * x_2 \stackrel{\alpha''}{\mapsto} x_1)$. The soundness of the constructions between symbolic heaps and SL-graphs is best illustrated by Lemma 10.

▶ **Lemma 10.** $(\Pi, \Sigma) \equiv_{\mathfrak{P}} (\mathsf{pure}(\mathsf{slg}(\Pi, \Sigma)), \mathsf{spatial}(\mathsf{slg}(\Pi, \Sigma)))$.

4.4 Relating memory states and deterministic SL-graphs

Given G_1 and G_2, we write $\mathfrak{f}_{G_1,G_2} : V_1 \rightarrow V_2$ to denote the map s.t. $\mathfrak{f}_{G_1,G_2}(v) \stackrel{\text{def}}{=} L_2(var(v))$. Without any further assumption, note that $\mathfrak{f}_{G_1,G_2}$ is not necessarily a precise homomorphism.

Given a $\mathfrak{P}$-memory state (s, h, ι) and a deterministic SL-graph $G = (A, V, \rightarrow, \Rightarrow, \overset{\neq}{\leftrightarrow}, L)$, we write $(s, h, \iota) \overset{\mathsf{lin}}{\approx} G$ whenever for all $v, v', v'' \in V$, if there are non-empty paths from v to v' and from v' to v'' in G, then for all variables x, x', x'' such that $L(x) = v$, $L(x') = v'$ and $L(x'') = v''$, one of the conditions below holds:

1. there is no non-empty sequence of memory cells in (s, h, ι) from $s(x)$ to $s(x'')$,
2. there is no non-empty sequence of memory cells in (s, h, ι) from $s(x'')$ to $s(x')$.

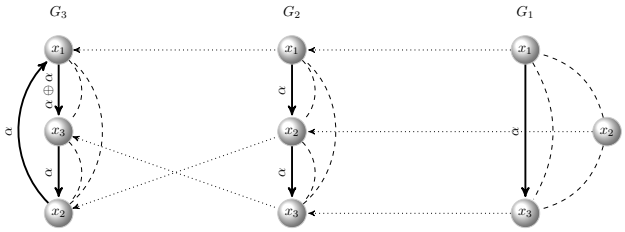
Roughly speaking, $(s, h, \iota) \overset{\mathsf{lin}}{\approx} G$ holds when (s, h, ι) respects the linearisation induced by the graphical part of G.

Below, we establish an equivalence between the existence of a precise homomorphism and the entailment $\models_{NI}$. A similar statement can be found in [6] but herein we deal with permissions and with the deterministic SL-graphs. This is a key result at the heart of our whole enterprise. An auxiliary definition is needed. Given a deterministic SL-graph G' and a symbolic heap (Π, Σ), we write $(\mathsf{pure}(G'), \mathsf{spatial}(G')) \models^{\mathsf{lin}} \Pi, \Sigma$ iff for all $\mathfrak{P}$-memory states (s, h, ι) such that $(s, h, \iota) \overset{\mathsf{lin}}{\approx} G$, if $(s, h, \iota) \models (\mathsf{pure}(G'), \mathsf{spatial}(G'))$, then $(s, h, \iota) \models \Pi, \Sigma$.

▶ **Lemma 11.** *Let G' be a deterministic SL-graph such that $(\mathsf{pure}(G'), \mathsf{spatial}(G'))$ is satisfiable. For all SL-graphs G, the statements below are equivalent:*

- *The map $\mathfrak{f}_{G,G'}$ is a precise homomorphism from G to G'.*
- $(\mathsf{pure}(G'), \mathsf{spatial}(G')) \models^{\mathsf{lin}} (\mathsf{pure}(G), \mathsf{spatial}(G))$.

Let us briefly explain below why in Lemma 11, $\models^{\mathsf{lin}}$ cannot be replaced by $\models$. Let us consider the deterministic SL-graphs G_1, G_2 and G_3 with respective permission formulae $\top$, $\top \wedge \mathsf{defined}(\alpha)$ and $\top \wedge \mathsf{defined}(\alpha) \wedge \mathsf{defined}(\alpha) \wedge \mathsf{defined}(\alpha)$ (all of them equivalent to $\top$).



Note that $\mathfrak{f}_{G_1,G_2}$ and $\mathfrak{f}_{G_2,G_3}$ are strongly precise homomorphisms and for $i \in \{1,2,3\}$, $(\mathsf{pure}(G_i), \mathsf{spatial}(G_i))$ is logically equivalent to (Π_i, Σ_i) from Example 2. However, not $(\mathsf{pure}(G_2), \mathsf{spatial}(G_2)) \models_{\mathfrak{P}_{\mathsf{Boy}}} (\mathsf{pure}(G_1), \mathsf{spatial}(G_1))$. Indeed, let us consider the $\mathfrak{P}_{\mathsf{Boy}}$-memory state (s,h,ι) defined in Section 2. We have $(s,h,\iota) \models (\mathsf{pure}(G_2), \mathsf{spatial}(G_2))$ and $(s,h,\iota) \not\models (\mathsf{pure}(G_1), \mathsf{spatial}(G_1))$. Actually, $(s,h,\iota) \overset{\mathsf{lin}}{\approx} G_2$ is false. By contrast, $(s,h,\iota) \overset{\mathsf{lin}}{\approx} G_3$. By Lemma 11, we conclude however that $(\mathsf{pure}(G_3), \mathsf{spatial}(G_3)) \models^{\mathsf{lin}} (\mathsf{pure}(G_2), \mathsf{spatial}(G_2))$ and $(\mathsf{pure}(G_2), \mathsf{spatial}(G_2)) \models^{\mathsf{lin}} (\mathsf{pure}(G_1), \mathsf{spatial}(G_1))$, which is sufficient for our needs.

4.5 Decision procedures modulo permission theories

Let us characterise non-entailment between two symbolic heaps in the non-intuitionistic setting (leading to entailment of symbolic heaps from Figure 3 with $G_2 = \mathtt{slg}(x_1 \neq x_3 \wedge x_1 \neq x_2 \wedge \alpha = \alpha', \mathsf{lseg}_{\alpha}(x_1,x_2) * \mathsf{lseg}_{\alpha'}(x_1,x_3) * x_2 \overset{\alpha''}{\mapsto} x_1)$. This induces a nondeterministic algorithm by guessing the appropriate G (see below). Such a guess could be formalised in a proof system, as done for a fragment in Section 3, but herein we focus on the characterisation. In Theorem 12 below, note the use of $\models_{NI}$ (instead of $\models^{\mathsf{lin}}$).

▶ **Theorem 12.** *Let* (Π,Σ), (Π',Σ') *be symbolic heaps s.t. neither* $\mathsf{slg}(\Pi,\Sigma)$ *nor* $\mathsf{slg}(\Pi',\Sigma')$ *is equal to* $\bot$. $(\Pi,\Sigma) \not\models_{NI} (\Pi',\Sigma')$ *iff there is a deterministic SL-graph* G *that satisfies:*
(SMALL) *For every* $v \overset{p}{\rightsquigarrow} v'$ *in* G, *the permission term* p *is a sum of at most* $|\Sigma|$ *terms from* $\mathsf{slg}(\Pi,\Sigma)$. *Moreover, the permission formula in* G *is precisely the permission part of* Π.
(SAT) $(\mathsf{pure}(G), \mathsf{spatial}(G))$ *is satisfiable.*
(PRE) $\mathfrak{f}_{\mathsf{slg}(\Pi,\Sigma),G}$ *is a strongly precise homomorphism.*
(NOTPRE) $\mathfrak{f}_{\mathsf{slg}(\Pi',\Sigma'),G}$ *is not a precise homomorphism.*

Here is our characterisation for satisfiability checking.

▶ **Theorem 13.** *Let* (Π,Σ) *be a symbolic heap so that* Σ *is* $\top$*-free and* $\mathtt{emp}$*-free and* $\mathsf{slg}(\Pi,\Sigma)$ *is not equal to* $\bot$. *Then,* (Π,Σ) *is satisfiable iff there is a deterministic SL-graph* G *such that (SMALL), (SAT) and (PRE) hold.*

The previous characterisations allow us to conclude to optimal complexity bounds.

▶ **Theorem 14.**
(I) SATSH($\mathfrak{P}_{\mathsf{Boy}}$) *is* NP*-complete and* ENTSH($\mathfrak{P}_{\mathsf{Boy}}$) *is* coNP*-complete.*
(II) *For any permission model* $\mathfrak{P}$ *such that* $\mathsf{width}(\mathfrak{P}) = \omega$ *and,* ENT($\mathfrak{P}$) *is in* coNP, SATSH($\mathfrak{P}$) *is* NP*-complete and* ENTSH($\mathfrak{P}$) *is* coNP*-complete.*

Currently, the permission terms do not allow constants other than $\mathbf{1}$ but for many permission models, constants can be easily added. For instance, in $\mathfrak{P}_{\mathsf{Boy}}$, one can deal with $\frac{3}{4}$ by using the variable α thanks to $(\alpha' \oplus \alpha' \oplus \alpha' \oplus \alpha' = \mathbf{1}) \wedge (\alpha = \alpha' \oplus \alpha' \oplus \alpha')$ (generalisation to other rational numbers is obvious). Of course, depending on the permission models in mind, constants should be either introduced in the language of permission terms (according to a specified encoding) or can be enforced directly in the language, as it is the case for $\mathfrak{P}_{\mathsf{Boy}}$.

5 Solving Permission Constraints

Herein, we review permission models introduced for separation logic and we classify these models according to the complexity of decision problems. In this process, we introduce Boolean permission models that generalise all existing permission models but the trivial model $\mathfrak{P}_{\mathbf{1}}$ and the fractional model $\mathfrak{P}_{\mathsf{Boy}}$ that admit PTIME decision problems. We give optimal complexity results for the satisfiability and entailment problems. This is motivated by Theorem 15 below, which is a consequence of the parameterised decision procedures from Section 4 that separate the reasoning on the memory shapes from the one on permissions.

▶ **Theorem 15.** *Let $\mathfrak{P}$ and $\mathfrak{P}'$ be permission models. (I)* $\mathrm{SAT}(\mathfrak{P}) = \mathrm{SAT}(\mathfrak{P}')$ *implies* $\mathrm{SATSH}(\mathfrak{P}) = \mathrm{SATSH}(\mathfrak{P}')$. *(II)* $\mathrm{ENT}(\mathfrak{P}) = \mathrm{ENT}(\mathfrak{P}')$ *implies* $\mathrm{ENTSH}(\mathfrak{P}) = \mathrm{ENTSH}(\mathfrak{P}')$.

We briefly review the permission models that have been introduced in the literature. The singleton model $\mathfrak{P}_{\mathbf{1}}$ presented in Section 2.2 is simply an artefact used to show that separation logic without permission is a special case of the general case. The fractional model $\mathfrak{P}_{\mathsf{Boy}}$ already presented in Section 2, is one of the most popular permission models and it enjoys nice complexity properties stated below.

▶ **Theorem 16.** $\mathrm{SAT}(\mathfrak{P}_{\mathsf{Boy}})$ *and* $\mathrm{ENT}(\mathfrak{P}_{\mathsf{Boy}})$ *are in* PTIME.

The PTIME bound is obtained by reduction to the satisfiability of a system of linear inequalities. Other classical permission models are :

- Bornat-Parkinson's permission model [4] with tokens is $\mathfrak{P}_{\mathsf{Tok}} = (P_{\mathfrak{P}_{\mathsf{Tok}}}, \mathbf{1}_{\mathfrak{P}_{\mathsf{Tok}}}, \oplus_{\mathfrak{P}_{\mathsf{Tok}}})$, where a permission $\pi \in P_{\mathfrak{P}_{\mathsf{Tok}}}$ is either a finite or a co-finite, non-empty subset of $\mathbb{N}$, $\mathbf{1}_{\mathfrak{P}_{\mathsf{Tok}}}$ is $\mathbb{N}$, and $\pi \oplus_{\mathfrak{P}_{\mathsf{Tok}}} \pi'$ is defined if $\pi \cap \pi' = \emptyset$ and is then equal to $\pi \cup \pi'$.
- Dockins-Hobor model, a.k.a binary shares [7], is $\mathfrak{P}_{\mathsf{Bin}} = ((\mathcal{T}(\mathcal{F})/\equiv) \setminus \{\mathbf{0}\}, \oplus, \mathbf{1})$ where $\mathcal{T}(\mathcal{F})$ is the set of closed terms constructed over the function symbols $\mathcal{F} = \{f : 2, \mathbf{0} : 0, \mathbf{1} : 0\}$, $\equiv$ is the least congruence such that $f(\mathbf{0}, \mathbf{0}) \equiv \mathbf{0}$ and $f(\mathbf{1}, \mathbf{1}) \equiv \mathbf{1}$, $\mathcal{T}(\mathcal{F})/\equiv$ denotes the quotient, and $\oplus$ is defined by $\mathbf{0} \oplus \mathbf{1} \equiv \mathbf{1} \oplus \mathbf{0} \equiv \mathbf{1}$, $\mathbf{0} \oplus \mathbf{0} \equiv \mathbf{0}$, $\mathbf{1} \oplus \mathbf{1}$ is undefined, and $f(\pi_1, \pi_2) \oplus f(\pi'_1, \pi'_2) \equiv f(\pi_1 \oplus \pi'_1, \pi_2 \oplus \pi'_2)$.

Note that in these two permission models a permission term $\alpha \oplus \alpha$ has no interpretation since the partial function $\oplus$ is not defined for identical elements. As a consequence, it holds for instance that $\mathsf{defined}(\alpha \oplus \alpha)$ is unsatisfiable and $\mathsf{lseg}_{\alpha\oplus\alpha}(x, y) \models_{\mathfrak{P}_{\mathsf{Tok}}} x \neq x$.

These two permission models are particular instances of what we call Boolean permission models, i.e. permission models defined on top of a given Boolean algebra, as explained below. Let $\mathbb{B} = (B_{\mathbb{B}}, \wedge_{\mathbb{B}}, \vee_{\mathbb{B}}, \top_{\mathbb{B}}, \bot_{\mathbb{B}}, \neg_{\mathbb{B}})$ be a Boolean algebra. The permission model $\mathfrak{P}_{\mathbb{B}}$ associated to $\mathbb{B}$ is $\mathfrak{P}_{\mathbb{B}} \stackrel{\text{def}}{=} (P_{\mathbb{B}}, \oplus_{\mathbb{B}}, \top_{\mathbb{B}})$ where $P_{\mathbb{B}} = B_{\mathbb{B}} \setminus \{\bot_{\mathbb{B}}\}$ and $\pi \oplus_{\mathbb{B}} \pi'$ is defined when $\pi \wedge_{\mathbb{B}} \pi' = \bot_{\mathbb{B}}$, and in that case $\pi \oplus_{\mathbb{B}} \pi' \stackrel{\text{def}}{=} \pi \vee_{\mathbb{B}} \pi'$. A permission model is **Boolean** if it is isomorphic to $\mathfrak{P}_{\mathbb{B}}$ for some Boolean algebra $\mathbb{B}$. Both $\mathfrak{P}_{\mathsf{Tok}}$ and $\mathfrak{P}_{\mathsf{Bin}}$ are Boolean, the first one through the Boolean algebra of finite or co-finite subsets of $\mathbb{N}$, the second one through the Boolean algebra of open-closed sets of $\{0,1\}^\omega$, see e.g. [7]. As stated below, Boolean permission models are canonical in some sense.

▶ **Lemma 17.** *Let A_L, A_R be permission formulae. Let $\mathfrak{P}$ be a Boolean permission model with at least two elements such that $A_L \models_{\mathfrak{P}} A_R$, and* $\mathsf{width}(\mathfrak{P}) \geq \mathit{card}(\mathtt{PVar}(A_L))$. *Then for all Boolean permission models $\mathfrak{P}'$, we have $A_L \models_{\mathfrak{P}'} A_R$.*

Lemma 17 entails $\mathrm{ENT}(\mathfrak{P}_{\mathsf{Tok}}) = \mathrm{ENT}(\mathfrak{P}_{\mathsf{Bin}})$. The case $\mathrm{card}(P_{\mathfrak{P}}) = 1$ cannot be added to Lemma 17 since $\top \models_{\mathfrak{P}_1} \alpha_1 = \alpha_2$ but $\top \not\models_{\mathfrak{P}_{\mathsf{Tok}}} \alpha_1 = \alpha_2$ with α_1 different from α_2. Note also that if we could express the atomicity of a permission, $\mathfrak{P}_{\mathsf{Tok}}$ and $\mathfrak{P}_{\mathsf{Bin}}$ could be distinguished.

▶ **Theorem 18.** $\mathrm{ENT}(\mathfrak{P}_{\mathsf{Tok}}) = \mathrm{ENT}(\mathfrak{P}_{\mathsf{Bin}})$ *and* $\mathrm{SAT}(\mathfrak{P}_{\mathsf{Tok}}) = \mathrm{SAT}(\mathfrak{P}_{\mathsf{Bin}})$.

From now on, we consider an arbitrary Boolean permission model $\mathfrak{P}_{\mathbb{B}}$ associated to a Boolean algebra $\mathbb{B}$ such that $\mathsf{width}(\mathfrak{P}_{\mathbb{B}}) = \omega$. Boolean permission models behave quite nicely and below we establish that their decision problems are in coNP.

▶ **Theorem 19.** *Let* $\mathfrak{P}_{\mathbb{B}}$ *be a Boolean permission model such that* $\mathsf{width}(\mathfrak{P}_{\mathbb{B}}) = \omega$. $\mathrm{SAT}(\mathfrak{P}_{\mathbb{B}})$ *is* NP*-complete and,* $\mathrm{ENT}(\mathfrak{P}_{\mathbb{B}})$ *is* coNP*-complete.*

A reduction from the NP-complete problem 1-in-3 SAT [14] gives the hardness result.

We give below the proof idea for $\mathrm{SAT}(\mathfrak{P}_{\mathbb{B}})$ is in NP. Actually, we can consider only permission terms equal to $\mathbf{1}$ and of the form $\bigoplus \alpha_i$ and atomic permission formulae of the form $\bigoplus \alpha_i = \bigoplus \alpha_j$, $\bigoplus \alpha_i \leq \bigoplus \alpha_j$, $\bigoplus \alpha_i = \mathbf{1}$ or $\mathsf{defined}(\bigoplus \alpha_i)$ and we can assume that each A contains a conjunct $\mathsf{defined}(p)$ for each permission term p. Let A be a permission formula built on $\alpha_1, \ldots, \alpha_n$. We introduce an arithmetical formula ψ_A such that A is satisfiable iff ψ_A is satisfiable. The Boolean/arithmetical variables of ψ_A taking their values in $\{0, 1\}$ are precisely $\mathtt{X}_1^1, \ldots, \mathtt{X}_1^n, \ldots, \mathtt{X}_n^1, \ldots, \mathtt{X}_n^n$. The formula ψ_A is a conjunction of formulae $\psi_1 \wedge \cdots \wedge \psi_n$ where each ψ_i is built on the variables $\mathtt{X}_1^i, \ldots, \mathtt{X}_n^i$. For each $i \in [1, n]$, we define $\mathfrak{t}^i(\alpha_j) \stackrel{\text{def}}{=} \mathtt{X}_j^i$, and $\mathfrak{t}^i(p)$ replaces each occurrence of α_j by $\mathfrak{t}^i(\alpha_j)$ and each occurrence of $\oplus$ by $+$, each occurrence of $\mathbf{1}$ by 1. Each ψ_i is a conjunction of constraints defined by: (a) for each $p = p'$ [resp. $p \leq p'$] in A, ψ_i contains $\mathfrak{t}^i(p) \leq \mathfrak{t}^i(p') \wedge \mathfrak{t}^i(p') \leq \mathfrak{t}^i(p)$ [resp. $\mathfrak{t}^i(p) \leq \mathfrak{t}^i(p')$] and (b) for each formula $\mathsf{defined}(p)$ in A, ψ_i contains $\mathfrak{t}^i(p) \leq 1$, and $\mathtt{X}_i^i = 1$ belongs to ψ_i. The next lemma relates A and ψ_A.

▶ **Lemma 20.** *A is satisfiable iff ψ_A is satisfiable.*

This lemma entails that $\mathrm{SAT}(\mathfrak{P}_{\mathbb{B}})$ is in NP.

6 Conclusion

Our results provide optimal complexity results about several standard permission models and are summarized by the following table. The algorithms can be implemented using any checker following the standard viewpoint for SMT solvers [2] for reasoning on permission. Besides, this work could be continued in several directions, for instance to consider enriched permission theories (e.g., adding inequalities between permission terms), permission models without infinite width, to allow existential quantifications or to design sequent-style proof systems for checking entailment based on our characterisations.

	$\mathfrak{P}_1$	$\mathfrak{P}_{\mathsf{Boy}}$	Boolean $\mathfrak{P}$, $\mathsf{width}(\mathfrak{P}) = \omega$ ($\mathfrak{P}_{\mathsf{Tok}}$, $\mathfrak{P}_{\mathsf{Bin}}$)
$\mathrm{SAT}(\mathfrak{P})$	in PTime	in PTime (Th. 16)	NP-C. (Th. 19)
$\mathrm{ENT}(\mathfrak{P})$	in PTime	in PTime (Th. 16)	coNP-C. (Th. 19)
$\mathrm{SATSH}(\mathfrak{P})$	in PTime [6]	NP-C. (Th. 9,14(I))	NP-C. (Th. 9,14(II))
$\mathrm{ENTSH}(\mathfrak{P})\backslash$ list pred.	in PTime [6]	in PTime (Th. 7)	coNP-C. (Th. 19,14(II))
$\mathrm{ENTSH}(\mathfrak{P})$	in PTime [6]	coNP-C. (Th. 9,14(I))	coNP-C. (Th. 9,14(II))

Acknowledgements. We thanks the anonymous referees for their remarks and suggestions.

References

1. C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *CAV'11*, volume 8606 of *LNCS*, pages 171–177. Springer, 2011.
2. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2008.
3. J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 97–109. Springer, 2004.
4. R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *POPL'05*, pages 259–270. ACM, 2005.
5. J. Boyland. Checking interference with fractional permissions. In *SAS'03*, number 2694 in LNCS, pages 55–72. Springer, 2003.
6. B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR'11*, volume 6901 of *LNCS*, pages 235–249, 2011.
7. R. Dockins, A. Hobor, and A.W. Appel. A fresh look at separation algebras and share accounting. In *APLAS'09*, volume 5904 of *LNCS*, pages 161–177. Springer, 2009.
8. D. Galmiche, D. Mery, and D. Pym. Resource tableaux (extended abstract). In *CSL'02*, volume 2471 of *LNCS*, pages 183–199. Springer, 2002.
9. C. Haase, S. Ishtiaq, J. Ouaknine, and M. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *CAV'13*, volume 8044 of *LNCS*, pages 790–795, 2013.
10. G. He, S. Qin, C. Luo, and W.N. Chin. Memory Usage Verification Using Hip/Sleek. In *ATVA'09*, number 5799 in LNCS, pages 166–181. Springer, 2009.
11. B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. Verifast: A powerful, sound, predictable, fast verifier for C and Java. In *NFM'11*, volume 6617 of *LNCS*, pages 41–55. Springer, 2011.
12. X. Bach Le, C. Gherghina, and A. Hobor. Decision procedures over sophisticated fractional permissions. In *APLAS'12*, pages 368–385, 2012.
13. J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.
14. Th. Schaefer. The complexity of satisfiability problems. In *STOC'78*, pages 216–226, 1978.
15. J. Villard, E. Lozes, and C. Calcagno. Tracking heaps that hop with Heap-Hop. In *TACAS'10*, volume 6015 of *LNCS*, pages 275–279. Springer, 2010.

Symmetric Synthesis*†

Rüdiger Ehlers[1] and Bernd Finkbeiner[2]

1 **University of Bremen, Bremen, Germany**
`ruediger.ehlers@uni-bremen.de`
2 **Saarland University, Saarbrücken, Germany**
`finkbeiner@cs.uni-saarland.de`

Abstract

We study the problem of determining whether a given temporal specification can be implemented by a symmetric system, i.e., a system composed from identical components. Symmetry is an important goal in the design of distributed systems, because systems that are composed from identical components are easier to build and maintain. We show that for the class of rotation-symmetric architectures, i.e., multi-process architectures where all processes have access to all system inputs, but see different rotations of the inputs, the symmetric synthesis problem is EXPTIME-complete in the number of processes. In architectures where the processes do not have access to all input variables, the symmetric synthesis problem becomes undecidable, even in cases where the standard distributed synthesis problem is decidable.

1998 ACM Subject Classification D.2.4 Formal Methods

Keywords and phrases Reactive Synthesis, Symmetry

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.26

1 Introduction

Many classical protocols and distributed systems are *symmetric*. This means that every process, independently of its identity, starts in the same initial state and follows the same set of transitions. Symmetric systems are easier to understand and maintain; especially in VLSI designs, which usually contain large numbers of identical components, this is a significant cost factor. Constructing symmetric systems is also a step towards building arbitrarily scalable systems [8, 2, 11].

There is a large body of results [1, 18, 5, 12, 26, 13] that deal with the question of which distributed systems need symmetry breaking and which do not. *Leader election* among the processes on a ring, for example, cannot be implemented symmetrically [1]; similarly, in resource-sharing problems, like the *Dining Philosophers*, the only way to avoid starvation is to break the symmetry [18].

Our goal is to automate this type of reasoning. Given a specification of a reactive system in temporal logic, we wish to automatically determine whether there exists a symmetric implementation. This is a refinement of the classic *distributed synthesis problem*, which asks whether a temporal specification has an implementation where the processes are arranged

* This work was partially supported by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative, by the German Research Foundation (DFG) within the program "Performance Guarantees for Computer Systems" and the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS), and by the European Research Council (ERC) Grant OSARES (No. 683300).

† A full version of the paper is available at [7], `https://arxiv.org/abs/1710.05633`

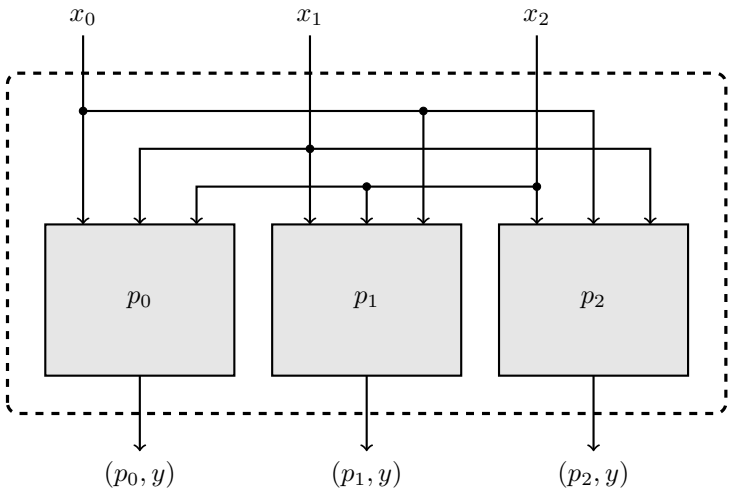


Figure 1 A simple rotation-symmetric architecture.

in a particular architecture. Distributed synthesis is well-studied [25, 21, 14, 15, 16, 9, 24]. However, the approach presented in this paper is the first to synthesize *symmetric* implementations. We consider *rotation-symmetric* system architectures. Rotation-symmetric architectures are multi-process architectures where all processes have access to all system inputs, but see different rotations of the inputs. Figure 1 shows a simple rotation-symmetric architecture. Rotation-symmetric architectures are suitable to reason about distributed systems that lack a central coordination process. They can, for example, model leader election scenarios and distributed traffic light controllers [6]. The fact that the processes obtain their input in different rotations is important: since all processes have the same implementation, they would otherwise also produce the same output. The synthesis problem for such systems could trivially be reduced to the standard synthesis problem by adding a constraint that the outputs are the same all the time.

We present an algorithm for the synthesis of symmetric systems in rotation-symmetric architectures from specifications in linear-time temporal logic (LTL). Most standard synthesis algorithms follow the automata-theoretic approach [22], whereby the given temporal formula is translated into a tree automaton that accepts exactly those computation trees that satisfy the formula. Hence, the specification is realizable if and only if the language of the automaton is non-empty. The synthesis algorithm then simply extracts some finite-state implementation from the language of the automaton. The situation is more difficult when we wish to decide the existence of a symmetric solution, because the language of the automaton may contain both computation trees that belong to symmetric implementations and computation trees that belong to asymmetric implementations. As we show in Section 4, symmetry is *not* a regular property: we therefore cannot check symmetry with a separate tree automaton or encode symmetry as a temporal logic formula and add it to the specification.

The key insight of our algorithm is that the paths in the computation trees produced by symmetric implementations are guaranteed to be invariant under rotations: if, in each position of two (finite or infinite) computation paths, the values of the input variables of the jth process in the first path correspond to the values of the input variables of the $((j + k) \bmod n)$th process, for some k, in the second path, then the values of the output variables of the jth process must also, in each position, correspond to the values of the output variables of the $((j + k) \bmod n)$th process (for all $0 \leq j < n$, where n is the number of

processes). Our algorithm exploits this observation to simplify the computation trees. Paths that are just rotations of each other are collapsed into a single representative. Computations in different processes that must lead to identical outputs are thus kept in the same path of the reduced tree; the paths only split when the symmetry is broken by some input. While symmetry is difficult to check on the original computation tree, it becomes a local condition on individual paths in the reduced tree: as long as the output never spontaneously introduces asymmetry, i.e., as long as every asymmetry in the output can be explained by a previous asymmetry in the input, the reduced tree can be expanded into a full computation tree that we know, by construction, to be symmetric.

As we show in Section 4, the running time of our synthesis algorithm is single-exponential in the number of processes. In Section 5, we show that our algorithm is asymptotically optimal: the problem is EXPTIME-complete in the number of processes. In Section 6, we study the extension of the synthesis problem to the case where the processes no longer have access to all variables. Here, our result is negative: under incomplete information, the symmetric synthesis problem is undecidable even for system architectures where the standard synthesis problem is decidable. This paper is based on previously unpublished results from the first author's PhD thesis [6], where also additional details of the presented results can be found. A full version of this paper with additional proofs is also available [7].

2 Preliminaries

A *reactive system* produces a valuation to the output propositions in some set AP^O and reads the values of the input propositions in some set AP^I in every step of its execution. The behavior of a reactive system can be described as a *computation tree* $\langle T, \tau \rangle$, where $T = (2^{\mathsf{AP}^I})^*$ is the set of tree nodes and $\tau : T \to 2^{\mathsf{AP}^O}$ labels every tree node t by the output propositions $\tau(t)$ that the system sets to **true** after having read t as its (prefix) input sequence.

A *trace* in a computation tree $\langle T, \tau \rangle$ is an infinite sequence $(\tau(\epsilon) \cup t_0)(\tau(t_0) \cup t_1)(\tau(t_0 t_1) \cup t_2)(\tau(t_0 t_1 t_2) \cup t_3) \ldots \in (2^{\mathsf{AP}^I \cup \mathsf{AP}^O})^\omega$. Given some language $L \subseteq (2^{\mathsf{AP}^I \cup \mathsf{AP}^O})^\omega$, *reactive synthesis* is the process of checking if there exists a computation tree $\langle T, \tau \rangle$ with $T = (2^{\mathsf{AP}^I})^*$ as node set such that every trace of $\langle T, \tau \rangle$ is in L. A classical logic to denote specification languages is linear temporal logic (LTL, [19]). LTL formulas for reactive system specifications are built according to the grammar

$$\varphi :== p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \mathsf{G}\,\varphi \mid \mathsf{F}\,\varphi \mid \varphi\,\mathcal{U}\,\varphi,$$

using the temporal operators G (*globally*), F (*eventually*), X (*next*), and $\mathcal{U}$ (*until*). All elements from AP^I and AP^O can be used as propositions p. A more formal definition of LTL is given in [19, 4].

For LTL specifications, it is known that if and only if there exists a computation tree all of whose traces satisfy a specification (i.e., the specification is *realizable*), there exists a *regular* such computation tree. A computation tree is regular if it has only finitely many different sub-trees. Given a computation tree $\langle T, \tau \rangle$, a tree $\langle T', \tau' \rangle$ is a *sub-tree* of $\langle T, \tau \rangle$ if and only if $T = T'$ and there exists a $\hat{t} \in T$ such that for every $t \in T$, we have $\tau'(t) = \tau(\hat{t}t)$. Regular computation trees can be translated to *finite-state machines* and implemented in hardware or software using a finite amount of memory. A tree language for some sets AP^I and AP^O is a subset of all trees $\langle T, \tau \rangle$ with $T = (2^{\mathsf{AP}^I})^*$ and $\tau : T \to 2^{\mathsf{AP}^O}$. A tree or word language is called *regular* if it can be recognized by some finite tree or word automaton (with a *Muller acceptance condition*, see [10] for details).

In *distributed synthesis*, we search for a distributed implementation of a finite state-machine. Given is an architecture that defines several *processes* and the *signals* that connect the processes among themselves and with the global input and output of the architecture. Starting from a specification over all signals, we search for implementations for all of the processes such that the computation tree *induced* by the process implementations and the architecture satisfies the specification. In the induced computation tree, all processes are executed at the same time and in parallel, using the usual parallel composition semantics.

It is known since the seminal work by Pneuli and Rosner [21] that not all architectures have a decidable distributed synthesis problem. Figure 2 depicts the *A0 architecture* that they defined as an example for an undecidable architecture. Finkbeiner and Schewe [9] later proved that the distributed synthesis problem is decidable if and only if there exists no *information fork* in the architecture. An information fork is a pair of processes that are incomparably informed, i.e., for which each of the processes has access to some global input that the other process cannot read. For a more formal definition of distributed synthesis, the interested reader is referred to [9].

A *Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, g)$ in which Q is a finite set of states, Σ is an input alphabet, $\Gamma \supseteq \Sigma$ is a (finite) tape alphabet, $\delta : Q \times \Gamma \to (Q \times \Gamma \times \{-1, 0, 1\})^2$ encodes the Turing machine transition function, $q_0 \in Q$ is an initial state, and g maps every state to its type, which can be *accepting*, *rejecting*, or *transient*. The δ function maps every state/tape content combination to exactly two possible successor state/tape content/tape motion combinations. For deterministic Turing machines, the two successor combinations are always the same. *Alternating Turing machines* [3] extend the non-deterministic Turing machines by partitioning the transient states into *universally branching* and *existentially branching states*. An (alternating) Turing machine accepts a word $w \in \Sigma^*$ if there exists an accepting run tree when starting in state q_0 with the tape empty except for a copy of w where the machine head starts on the first character of w. In all universal states, the Turing machine execution must be accepting for both possible transitions.

We assume that the *modulo function* always returns a non-negative number, such that, e.g., $-13 \mod 5 = 2$.

3 The Symmetric Synthesis Problem

We consider distributed reactive synthesis problems in which all processes share the same implementation. A process has an *interface* $\mathcal{N} = (\mathsf{AP}^I, \mathsf{AP}^O)$ with the local input proposition set AP^I and a local output proposition set AP^O. The connections between the processes are described in an *architecture*.

▶ **Definition 1** (Symmetric architecture). Given an interface $\mathcal{N} = (\mathsf{AP}^I, \mathsf{AP}^O)$, a *symmetric architecture* over $\mathcal{N}$ is a tuple $\mathcal{E} = (S, P, \mathsf{AP}^I_G, E^{in}, E^{out})$ with:

- the set of (internal) signals S,
- the *process set* P,
- the *global input signal set* AP^I_G,
- the *input edge function* $E^{in} : (P \times \mathsf{AP}^I) \to (S \cup \mathsf{AP}^I_G)$, and
- the *output edge function* $E^{out} : (P \times \mathsf{AP}^O) \to S$.

As an example, the architecture given in the right part of Figure 2 hosts processes with the interface $\mathcal{N} = (\{a\}, \{b\})$ and has the components $S = \{y, z\}$, $P = \{0, 1\}$, $\mathsf{AP}^I_G = \{x\}$, $E^{in} = \{(0, a) \mapsto x, (1, a) \mapsto y\}$, and $E^{out} = \{(0, b) \mapsto y, (1, b) \mapsto z\}$. We only consider architectures in which every internal signal is written to by exactly one local output of one process. Given a FSM for a process with an interface $\mathcal{N}$ and an architecture $\mathcal{E} = (S, P, \mathsf{AP}^I_G, E^{in}, E^{out})$ over

$\mathcal{N}$, we can construct an FSM with AP_G^I as input proposition set and S as output proposition set that implements the behavior of the complete architecture when using the FSM as process implementation. Without loss of generality, we use the standard synchronous composition semantics to do so. We define the symmetric synthesis problem as follows:

▶ **Definition 2.** Given an interface $\mathcal{N} = (\mathsf{AP}^I, \mathsf{AP}^O)$, an architecture $\mathcal{E} = (S, P, \mathsf{AP}_G^I, E^{in}, E^{out})$, and a specification φ over the propositions $\mathsf{AP}_G^I \cup S$, *the symmetric synthesis problem* is to check if an FSM implementation $\mathcal{F}$ with the input proposition set AP^I and output proposition set AP^O exists such that the FSM obtained by plugging $\mathcal{F}$ into $\mathcal{E}$ satisfies φ. In case of a positive answer, we also want to obtain $\mathcal{F}$.

4 Rotation-Symmetric Synthesis

Many symmetric architectures found in practice consist of a ring of processes, all of which read all the input to the overall system. A slight generalization of this architecture shape is the class of *rotation-symmetric architectures*.

▶ **Definition 3.** A symmetric architecture $\mathcal{E} = (S, P, \mathsf{AP}_G^I, E^{in}, E^{out})$ over the interface $\mathcal{N} = (\mathsf{AP}^I, \mathsf{AP}^O)$ with n processes is called *rotation-symmetric* if and only if there exists a *local designated proposition set* AP_L^I for every process instance such that the following conditions hold:

- $\mathsf{AP}_G^I = \mathsf{AP}^I = \mathsf{AP}_L^I \times \{0, \dots, n-1\}$ and $P = \{p_0, \dots, p_{n-1}\}$.
- $S = \mathsf{AP}^O \times \{0, \dots, n-1\}$
- for every $p_i \in P$, every $x \in \mathsf{AP}_L^I$, and every $j \in \{0, \dots, n-1\}$, we have $E^{in}(p_i, (x, j)) = (x, (j - i) \mod n)$, and
- for every $x \in \mathsf{AP}^O$ and $p_i \in P$, we have $E^{out}(p_i, x) = (x, i)$.

We show in this section that the symmetric synthesis problem for rotation-symmetric architectures and linear-time temporal logic (LTL) is decidable.

The key observation that we use to prove decidability is that the computation trees that characterize the input/output behavior of a process implementation plugged into a rotation-symmetric architecture have a useful property that we call the *symmetry* property. While this property is non-regular and thus cannot be encoded into the specification (Lemma 6), we show how to decompose it into two sub-properties, one of which is regular. The other one is still non-regular, but has the advantage that we can enforce it in a synthesis process by post-processing the computation tree obtained from a synthesis procedure to contain only rotations of the computation tree paths along so-called *normalized inputs*. Since every tree with the symmetry property is left unaltered by this step and we also describe how to ensure that the result of the post-processing step is guaranteed to be a correct solution, this approach is sound and complete.

We assume some fixed rotation-symmetric architecture $\mathcal{E} = (S, P, \mathsf{AP}_G^I, E^{in}, E^{out})$ over some local process interface $(\mathsf{AP}_L^I, \mathsf{AP}^O)$ to be given, define $\mathcal{I} = 2^{AP_G^I}$ to denote the global input alphabet to all processes, while $\mathcal{O} = 2^{\{\mathsf{AP}^O \times \{0,\dots,n-1\}\}}$ denotes the global output. The local output of one process is given as $O = 2^{\mathsf{AP}^O}$.

The following rotation function will become useful in the analysis below. Let $U = 2^{\mathsf{AP} \times \{0,\dots,n-1\}}$ for some other set AP. We define a *rotation operator* $\mathrm{rot} : U \times \mathbb{Z} \to U$ with $\mathrm{rot}(u, k) = \{(p, (j+k) \bmod n) \mid (p, j) \in u\}$ for every $u \in U$ and $k \in \mathbb{Z}$. Furthermore, we extend the rot function to LTL formulas and define $\mathrm{rot}(\psi, k)$ for an LTL formula ψ over the set of propositions $\mathsf{AP} \times \{0, \dots, n-1\}$ and $k \in \mathbb{Z}$ to be ψ with all atomic propositions (p, j) replaced by $(p, (j+k) \bmod n)$ for $p \in \mathsf{AP}$, $j \in \mathbb{Z}$. For clarity, when dealing with the rot function for some

set $U = 2^{\mathsf{AP}\times\{0,\dots,n-1\}}$, we often partition the elements of $\mathsf{AP}\times\{0,\dots,n-1\}$ by their process indices and for example write $(X_0,\dots,X_{n-1})$ instead of $(X_0\times\{0\})\cup\ldots\cup(X_{n-1}\times\{n-1\})$ for $X_0,\dots,X_{n-1}\subseteq\mathsf{AP}$. The rotation function is extended to sequences of elements in U by rotating the individual sequence items.

▶ **Definition 4** (Symmetry property)**.** Given a tree $\langle T,\tau\rangle$ over $T=\mathcal{I}^*$ and $\tau: T\to\mathcal{O}$, we say that the tree has the *symmetry property* if for each $t\in T$ and $0\le i<n$, $\tau(\mathrm{rot}(t,i)) = \mathrm{rot}(\tau(t),i)$.

▶ **Lemma 5** (Symmetry lemma)**.** *The set of regular trees having the symmetry property is precisely the same as the set of trees that are induced by a rotation-symmetric architecture for some process implementation.*

A proof of the lemma can be found in the full version of this paper [7]. The symmetry property is not a regular tree property, and hence cannot be encoded into a tree or word automaton.

▶ **Lemma 6.** *The set of symmetric computation trees for the two-process rotation-symmetric architecture with process interface* $\mathcal{N} = (\mathsf{AP}_L^I\times\{0,1\},\mathsf{AP}^O)$ *and* $\mathsf{AP}_L^I=\{i\}$ *and* $\mathsf{AP}^O=\{o\}$ *is not a regular tree language.*

Proof. For a proof by contradiction, suppose that the set of symmetric computation trees is regular. The language includes a tree with the symmetry property in which the node labels on the path $(\emptyset,\{i\})^*$ and, symmetrically, on the path $(\{i\},\emptyset)^*$ form the sequence $l=(\emptyset,\emptyset)^1(\{o\},\{o\})(\emptyset,\emptyset)^2(\{o\},\{o\})\ldots$, i.e., the length of the $(\emptyset,\emptyset)$-sequences grows according to the distance to the root. According to the pumping lemma for regular tree languages, however, the sequence l can be partitioned into $l=u\cdot v\cdot w$, such that, for every $k>0$, there exists a tree in the language where the label sequence on $(\emptyset,\{i\})^*$ is $l=u\cdot v^k\cdot w$, while the label sequence on $(\{i\},\emptyset)^*$ is still l. Clearly, these trees are not symmetric. ◀

Since the symmetry property is non-regular, we need to alter the synthesis process itself to account for it. In order to synthesize an implementation for *one* process, we synthesize implementations for *all* processes together. These only need to work correctly on *normalized input sequences* $t\in\mathcal{I}^*$. An input sequence is *normalized* if $\min_i \mathrm{rot}(t,i)=t$, where the min function uses the lexicographic ordering over the strings in $\mathcal{I}^*$. For the ordering of the elements in $\mathcal{I}$, we consider the lexicographic ordering of their tuple representation. For example, we have $(0,1,0)<(0,1,1)$ and $(0,1,0)<(1,0,0)$ for a three-process architecture. A tree with the symmetry property is fully determined by the labels along normalized input sequences, as for every non-normalized input sequence $t'\in\mathcal{I}^*$, we have $\tau(t')=\mathrm{rot}(\tau(t),i)$ for every i such that $t'=\mathrm{rot}(t,i)$.

When only considering the normalized input sequences during synthesis, we can take the computation tree for all processes in the architecture together and complete it by filling all other tree labels with rotations of the tree labels along normalized inputs. We call the resulting tree its *symmetric completion*. If afterwards, we have $\tau(\mathrm{rot}(t,i))=\mathrm{rot}(\tau(t),i)$ for all $t\in\mathcal{I}^*$ and $i\in\mathbb{N}$, then the symmetry lemma guarantees that the resulting tree is induced by some process instantiated in a rotation-symmetric architecture. So if we can guarantee that (1) $\tau(\mathrm{rot}(t,i))=\mathrm{rot}(\tau(t),i)$ is actually the case for all normalized t and $i\in\mathbb{N}$ and (2) that the symmetric completion of the tree satisfies the specification along all paths, then we can obtain a correct process implementation by synthesizing a computation tree for the complete architecture. Our construction for symmetric synthesis consist of these two components, which we describe in more detail below.

4.1 Ensuring Symmetric Completability

Not every $\mathcal{O}$-labeled computation tree can easily be made symmetric by replacing the tree labels for non-normalized input sequences. Take for example a tree $\langle T, \tau \rangle$ for the architecture given in Figure 1 with $\tau(\epsilon) = (\emptyset, \emptyset, \{y\})$. Since the output of the processes is initially different, this means that they cannot have the same implementation. We show in this section that detecting such cases is simple, and the formalization of the observation is a regular property that can be easily encoded into LTL.

▶ **Definition 7.** Let AP be some set, and $P = \{p_0, \dots, p_{n-1}\}$ be a list of process identifiers. For every $x \subseteq (\mathsf{AP} \times \{0, \dots, n-1\})$ and $w = w_0 w_1 w_2 \dots w_l \in (2^{\mathsf{AP} \times \{0,\dots,n-1\}})^*$, we define

$$\begin{aligned} \mathrm{rep}(x) &= |\{j \in \{0, \dots, n-1\} \mid \mathrm{rot}(x, j) = x\}| \\ \mathrm{rep}_S(\epsilon) &= n \\ \mathrm{rep}_S(w) &= \gcd(\mathrm{rep}_S(w_0 \dots w_{n-1}), \mathrm{rep}(w_n)), \end{aligned}$$

where gcd denotes the *greatest common divisor* function.

For some word $t \in \mathcal{I}^*$, $\mathrm{rep}_S(t)$ represents how many different rotations in $\{0, \dots, n-1\}$ of t exist that map the word to itself.

▶ **Lemma 8** (Second symmetry lemma). *Let $\langle T, \tau \rangle$ be a computation tree with $T = \mathcal{I}^*$ and $\tau : T \to \mathcal{O}$ for which for every $t \in T$, we have that $\mathrm{rep}_S(t) \mid \mathrm{rep}_S(\tau(t))$ (where the $\mid$ symbol refers to division without remainder). The unique symmetric completion of $\langle T, \tau \rangle$ has the symmetry property. Furthermore, if $\langle T, \tau \rangle$ is regular, then so is its unique symmetric completion.*

By the second symmetry lemma, it suffices for a computation tree to have $\mathrm{rep}_S(t) \mid \mathrm{rep}_S(\tau(t))$ for all $t \in T$ to ensure that the symmetric completion of the tree has the symmetry property. We can encode this requirement in LTL as

$$\varphi_{outcond} = \bigwedge_{d \in \{1,\dots,n\}, d \mid n} \neg(\mathrm{sym}(\mathcal{I}, d, n) \,\mathcal{U}\, \neg \mathrm{sym}(\mathcal{O}, d, n))$$

for the function

$$\mathrm{sym}(\mathsf{AP}, d, n) = \bigwedge_{a \in \mathsf{AP}, j \in \{0,\dots,n-1\}} (a, j) \leftrightarrow (a, j + \frac{n}{d})$$

that encodes, for each $i \subseteq \mathsf{AP} \times \{0, \dots, n-1\}$ whether $d \mid \mathrm{rep}(i)$ (for $d \in \mathbb{N}$ with $d \mid n$).

4.2 Ensuring That the Tree Completion Satisfies the Specification

If we have a computation tree $\langle T, \tau \rangle$ all of whose traces satisfy some linear-time specification φ, this does not imply that its rotation-symmetric completion satisfies φ as well. If all traces of $\langle T, \tau \rangle$ however satisfy $\varphi \wedge \mathrm{rot}(\varphi, 1) \wedge \dots, \wedge \mathrm{rot}(\varphi, n-1)$, then since we know that every infinite trace in the rotation-symmetric completion is a rotation of a trace in the original tree by some value $i \in \mathbb{N}$, we know that the rotation-symmetric completion also satisfies φ along every trace. So if we synthesize a tree for $\varphi' = \varphi \wedge \mathrm{rot}(\varphi, 1) \wedge \dots \wedge \mathrm{rot}(\varphi, n-1)$ as specification instead of φ, taking the rotation-symmetric completion maintains φ.

Note that strengthening φ to φ' comes without loss of generality if we are interested in rotation-symmetric implementations. By the symmetry property, if the tree $\langle T, \tau \rangle$ induced

by a rotation-symmetric architecture and a process implementation satisfies φ, then it also satisfies $\mathrm{rot}(\varphi, i)$ for all $i \in \mathbb{N}$ as every rotation of every trace in the tree is also a trace in the tree. Hence, to satisfy φ, it also needs to satisfy $\mathrm{rot}(\varphi, i)$ as otherwise we could take a trace not satisfying $\mathrm{rot}(\varphi, i)$, rotate it by $-i$, and obtain a trace that does not satisfy φ.

4.3 Putting Everything Together

Using the concepts defined above, we are now ready to tie them together to a complete synthesis process. We start with a specification φ over the architecture input propositions AP_G^I and the output proposition set $\mathsf{AP}^O \times \{0, \ldots, n-1\}$ for $|P| = n$.

1. We modify the specification φ to $\varphi' = \varphi \wedge \mathrm{rot}(\varphi, 1) \wedge \ldots \wedge \mathrm{rot}(\varphi, n-1)$.
2. We modify φ' to $\varphi'' = \varphi' \wedge \varphi_{outcond}$ (as described in Section 4.2).
3. We synthesize a regular tree $\langle T, \tau \rangle$ that satisfies φ'' along all paths using a classical reactive synthesis procedure. If there is no such tree, the specification is unrealizable.
4. If a regular computation tree $\langle T, \tau \rangle$ is found, we replace every label along non-normalized directions by rotations of τ's labels along normalized directions to get a tree $\langle T', \tau' \rangle$ with the symmetry property.
5. We cut off the labels of τ' except for the output of the first process in the architecture. The resulting (regular) tree is the synthesized process implementation.

▶ **Proposition 9.** *The above synthesis process from LTL has a complexity that is 2EXPTIME in the length of the specification and exponential-time in the number of processes.*

Proof. We use the automata-theoretic approach to reactive system synthesis from [17, 24] and the concepts defined in these works. We start by translating the specification to a universal co-Büchi word (UCW) automaton, which is of size $2^{O(|\varphi|)}$ in the size of the specification. As UCWs do not blow up under conjunction, executing step 1 from the construction above leads to an automaton of size $n \cdot 2^{O(|\varphi|)}$. A deterministic automaton for the added property in step 2 can be built with at most n states, so executing step 2 leads to at most n additional states, and we obtain an automaton with $n + n \cdot 2^{O(|\varphi|)} = n \cdot 2^{O(|\varphi|)}$ many states. The *bounded synthesis* approach works with specifications given as co-Büchi word automata [24] and takes time exponential in the number of states of the automaton. The overall time complexity so far is thus 2EXPTIME in $|\varphi|$ and exponential in n. Step 4 leads to a blow-up of at most a factor of n^2 and can be done in time polynomial in the number of states in the synthesized finite-state machine (whose size is proportional to the time complexity of the synthesis procedure executed in the previous step). Step 5 is simple and takes time linear in the size of the FSM. ◀

Note that even though the construction above discards all non-normalized parts of the synthesized computation tree, asking the synthesis algorithm to nevertheless synthesize these parts according to the specification comes without loss of generality, as trees with the symmetry property (which we are actually searching for) fulfill φ'' along all paths if all of their paths satisfy φ. So the synthesis process does not report spurious unrealizability.

5 Rotation-Symmetric Synthesis – Complexity

The symmetric synthesis construction from the previous section has a time complexity that is doubly-exponential in the length of the specification and singly-exponential in the number of processes. We want to show in this section that this matches the complexity of the problem by giving a corresponding hardness result. The 2EXPTIME-hardness in the specification

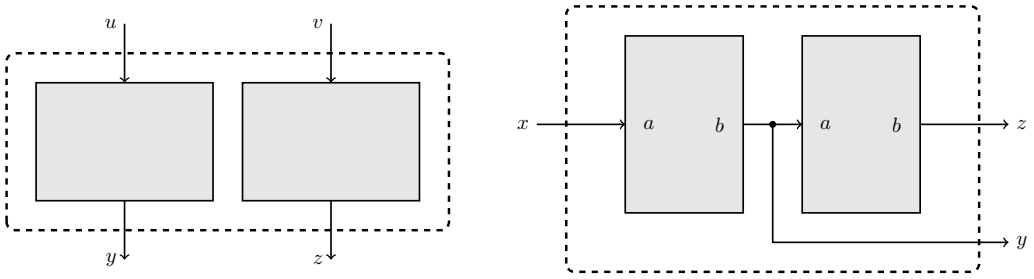


Figure 2 System architectures with undecidable synthesis problems. On the left: architecture A0, as defined by Pnueli and Rosner [21]; on the right: the symmetric architecture S0. The distributed synthesis problem of A0 and the symmetric synthesis problem of S0 are undecidable.

length is inherited from the complexity of LTL synthesis [20]. For the EXPTIME complexity in the number of processes, we provide the following result:

▶ **Lemma 10.** *Given an $f(k)$-space bounded alternating Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, g)$, we can reduce the acceptance of a word $w \in \Sigma^k$ by M to the symmetric realizability problem of $n = f(k)$ processes with a specification in LTL of size polynomial in $|Q| \cdot |\Gamma| \cdot |w|$.*

Proof. We build a specification that requires the processes to output the Turing tape configuration along an execution of the machine. The specification is realizable if and only if the Turing machine does **not** accept the word. Every process outputs the value of one Turing tape cell and if the tape head is at the cell, also the state of the Turing machine. There are n input signals to the architecture, and when the processes start, the left-most local input signals of the processes is used to tell one or more processes that the Turing tape computation should start at that cell with the tape head being initially there (with w as the initial tape content). To account for the rotation-symmetry, the processes output not only the tape content and tape head position, but also the current boundaries of the tape. The specification is modeled such that if start and end markers collide, the simulation of the Turing machine can stop.

The specification also includes conjuncts that require all processes together to simulate the Turing machine computation correctly and to never reach an accepting state. Whenever the alternating Turing machine branches universally, the left-most local process input signal is used to select which successor state is picked. In case of existential branching, the processes can decide which successor state to pick. Enforcing the specification to be realizable if and only if the word w is **not** accepted by the Turing machine helps with taking care of the diverging computations of the Turing machine and those computations that exceed the space bound. Both count as non-accepting in the definition of space-bounded Turing machines. Since these runs never visit accepting states and/or permit the simulation to stop, they are allowed to be simulated by a synthesized implementation.

The specification can be written with size polynomial in $|Q| \cdot |\Gamma| \cdot |w|$ as we only need to define the specification for one process. By the symmetry of the architecture, the other processes have to fulfill it as well. ◀

A more detailed proof can be found in the full version of this paper [7].

▶ **Corollary 11.** *The rotation-symmetric realizability problem (for LTL) has a time complexity that is exponential in the number of processes.*

Proof. Given the question whether a word $w = w_0 \dots w_{k-1}$ is in the language defined by some $(c+1)$-EXPTIME = (c)-AEXPSPACE problem for some $c \in \mathbb{N}$, we can reduce it to the

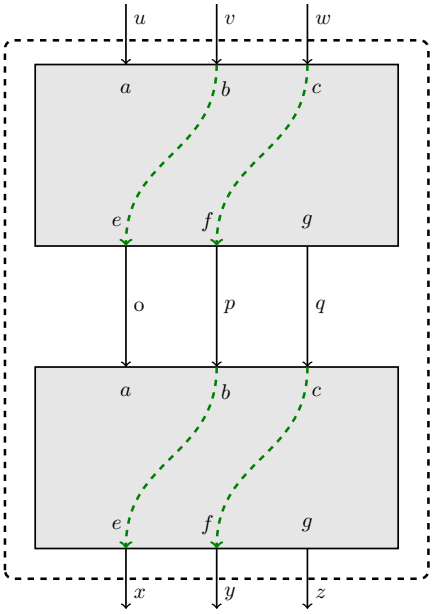


Figure 3 Symmetric architecture S2. The symmetric synthesis problem for S2 is undecidable. The dashed arrows in the process boxes show how the specification given in the proof of Lemma 13 requires the processes to forward the local input streams.

symmetric realizability problem for an LTL specification of length polynomial in k and with a number of processes that is (c)-exponential in k. Since by the space hierarchy theorem [23], the (c)-EXPTIME hierarchy is strict for increasing c, we can conclude that in general, we cannot solve the symmetric realizability problem faster than in time exponential in the number of components. ◀

6 The General Case – Undecidability

The synthesis problem for standard, not necessarily symmetric, distributed systems is decidable as long as the processes can be ordered with respect to their relative knowledge about the system inputs [9]. The problem becomes undecidable as soon as it contains an information fork, i.e., a pair of processes with incomparable knowledge. The simplest such architecture is Pnueli and Rosner's A0 architecture [21], shown on the left in Fig. 2. In this section, we show that for symmetric synthesis, even architectures without information forks, such as the S0 architecture shown on the right in Fig. 2, are undecidable. Our proof is based on Pnueli and Rosner's undecidability argument for A0:

▶ **Lemma 12** ([21]). *For a given Turing machine M, there exists an LTL formula ψ that is realizable in the distributed architecture A0 if and only if M halts and such that the two processes of the unique implementation of M sequentially output binary encodings of the configurations of the Turing machine on y (or z, respectively) upon the first* **true** *value on the input u (or v, respectively).*

Because of the undecidability of the halting problem, Lemma 12 means that the distributed synthesis problem of architecture A0 is undecidable. We prove the undecidability of the symmetric synthesis problem of architecture S0 in two steps. First, we establish the undecidability of the larger architecture S2, depicted in Figure 3, by showing that the

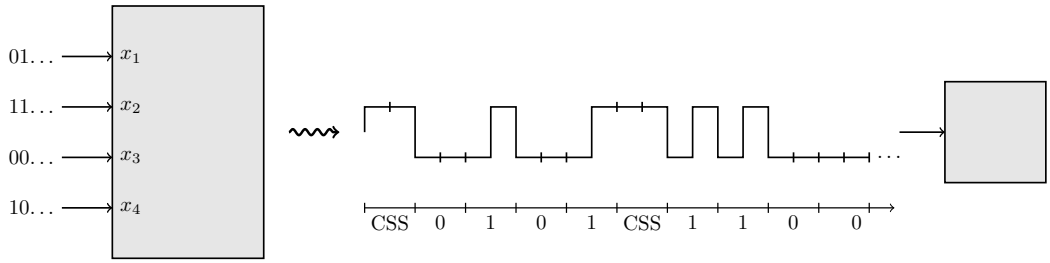


Figure 4 An example for compressing a word with $|\mathsf{AP}| = 4$.

realizability of ψ in A0 can be reduced to the symmetric realizability of an LTL formula over S2; in the second step, we encode the synthesis problem of S0 into the synthesis problem of S2 and thus establish that the synthesis problem for the simpler architecture S0 is undecidable as well.

▶ **Lemma 13.** *The symmetric synthesis problem for architecture S2 is undecidable.*

Proof. We show that there exists an implementation for the specification ψ in the A0 architecture if and only if there exists a joint implementation for the two processes in the S2 architecture that satisfies $\psi' = \psi_d \wedge \mathsf{G}(v \leftrightarrow \mathsf{X}o) \wedge \mathsf{G}(w \leftrightarrow \mathsf{X}p)$, where ψ_d results from prefixing all occurrences of the signals y and z in ψ with a next-time operator.

The results of the two synthesis problems can be translated into each other. A distributed implementation of ψ over A0 is necessarily symmetric: both processes output the same bitstream when reading a **true** value as their local input for the first time. To obtain an implementation for S2, we simulate the process with input a and use g as the local output. Additionally, we copy all values from b to e, and c to f.

Conversely, an implementation found by the symmetric synthesis of S2 provides an implementation of ψ in A0. The key property of the architecture S2 is that the process does not know if the local input b is the (delayed) a input to the other process, or if its c input is the (Turing machine tape) output of the other process. Thus, it cannot find out if it is the top process or the bottom process in the architecture and must prevent violating the specification in either case. A more detailed proof is given in the full version of this paper [7]. ◀

In order to reduce the symmetric synthesis problem of S2 to the symmetric synthesis problem of S0, we introduce compression functions that time-share multiple signals of S2 into a single signal in S0.

Let AP be a set of signals. We call a function $f : (2^{\mathsf{AP}})^\omega \to (2^{\{\chi\}})^\omega$ for some Boolean variable χ a *compression function* if f is injective. We call a function f' that maps a specification over the signal set AP to a different specification over the signal set $\{\chi\}$ the *adjunct compression function* to f if for all $w \in (2^{\mathsf{AP}})^\omega$ and specifications ψ over AP, we have that $w \models \psi$ if and only if $f(w) \models f'(\psi)$.

In the full version of this paper [7], we give such a pair of compression functions for LTL. The compression mechanism is illustrated in Figure 4. One clock cycle in the four-bit-per-character version of a word is spread to 10 computation cycles in the one-bit-per-character version of the word. Every 10 cycles, the 2-cycle *character start sequence* (CSS) $\{\chi\}\{\chi\}$ is instantiated, followed by four two-cycle slots for every signal in AP. Note that the construction ensures that whenever we have $\{\chi\}\{\chi\}\emptyset$ as a part in a compressed word, then we know that a character start sequence begins on the first occurrence of $\{\chi\}$ in this part.

▶ **Theorem 14.** *The symmetric synthesis problem for architecture S0 is undecidable.*

Proof. In order to reduce the symmetric synthesis problem of architecture $S0$ to the symmetric synthesis problem of architecture S2, we compress u, v, w into signal x; o, p, q into signal y; and x, y, z into signal z. A more detailed proof is given in the full version of this paper [7]. ◀

7 Conclusions

In this paper, we have studied the problem of synthesizing symmetric systems. Our new synthesis algorithm is a useful tool in the development of distributed algorithms, because it checks automatically if certain properties in a design problem require symmetry breaking.

Our algorithm synthesizes implementations of rotation-symmetric architectures, i.e., architectures where the processes observe all inputs. The undecidability result for the architecture S0 indicates that it is impossible to extend the synthesis algorithm to architectures where the processes no longer have access to all inputs. A promising direction of research, however, is to use our results to extend existing *semi-algorithms* for synthesis under incomplete information to such symmetric architectures. An example for such an approach is *bounded synthesis* [24], which determines if there exists an implementation with at most n states, where n is a given bound. The specification is translated into a universal co-Büchi automaton, which is then, together with the bound n, encoded into a *satisfiability modulo theory* problem. To ensure correctness under incomplete information, constraints are added that ensure that if a process cannot distinguish two inputs, it transitions to the same successor state. Similarly, for symmetric synthesis, constraints can be added that ensure that the outputs of the individual processes are identical in states that are indistinguishable for them.

Algorithms for symmetric synthesis procedures also offer a new perspective on the problem of synthesizing arbitrarily scalable (i. e. *parametric*) systems. Due to the undecidability of the problem, only very limited solutions to this problem have been found so far. For example, Jacobs and Bloem [11] tackle the case of asynchronous processes with local input in a ring architecture and use the bounded synthesis approach mentioned above. Emerson and Srinivasan [8] present a solution for a multi-process version of a small subset of the temporal logic CTL while Attie and Emerson [2] give a different solution allowing a bigger subset of CTL but only guaranteeing correctness of the solution if certain other conditions are fulfilled, like the dead-lock freeness of the solution produced. In such a setting, symmetric synthesis can be used to detect specifications that are unrealizable even for small system sizes – if there is no solution for a fixed number of processes n, then there is certainly none for scalable systems as well.

References

1 Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Twelfth Annual ACM Symposium on Theory of Computing (STOC)*, pages 82–93, 1980.

2 Paul C. Attie and E. Allen Emerson. Synthesis of concurrent systems with many similar processes. *ACM Trans. Program. Lang. Syst.*, 20(1):51–115, 1998. doi:10.1145/271510.271519.

3 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

4 E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

5 Shimon Cohen, Daniel J. Lehmann, and Amir Pnueli. Symmetric and economical solutions to the mutual exclusion problem in a distributed system. *Theor. Comput. Sci.*, 34:215–225, 1984.

6 Rüdiger Ehlers. *Symmetric and efficient synthesis*. PhD thesis, Saarland University, 2013. URL: `http://scidok.sulb.uni-saarland.de/volltexte/2013/5607/`.

7 Rüdiger Ehlers and Bernd Finkbeiner. Symmetric synthesis. *ArXiV/CoRR*, 1710.05633, 2017. Full version of this paper.

8 E. Allen Emerson and Jai Srinivasan. A decidable temporal logic to reason about many processes. In *Proc. PODC*, pages 233–246, 1990.

9 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proc. LICS*, pages 321–330, 2005.

10 Jörg Flum, Erich Grädel, and Thomas Wilke, editors. *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*. Amsterdam University Press, 2008.

11 Swen Jacobs and Roderick Bloem. Parameterized synthesis. *Logical Methods in Computer Science*, 10(1), 2014. `doi:10.2168/LMCS-10(1:12)2014`.

12 Ralph E. Johnson and Fred B. Schneider. Symmetry and similarity in distributed systems. In *Proc. PODC*, pages 13–22. ACM, 1985.

13 Evangelos Kranakis. Invited talk: Symmetry and computability in anonymous networks. In Nicola Santoro and Paul G. Spirakis, editors, *Proc. SIROCCO*, pages 1–16. Carleton Scientific, 1996.

14 Orna Kupferman and Moshe Y. Vardi. Synthesis with incomplete information. In *Proc. ICTL*, 1997.

15 Orna Kupferman and Moshe Y. Vardi. μ-calculus synthesis. In *Proc. MFCS*, pages 497–507, 2000.

16 Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS 2001)*, July 2001.

17 Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542. IEEE, 2005.

18 Daniel J. Lehmann and Michael O. Rabin. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In *Proc. POPL*, 1981.

19 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

20 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.

21 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, volume II, pages 746–757. IEEE, 1990.

22 Michael O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, 1972.

23 Desh Ranjan, Richard Chang, and Juris Hartmanis. Space bounded computations: Review and new separation results. *Theor. Comput. Sci.*, 80(2):289–302, 1991. `doi:10.1016/0304-3975(91)90391-E`.

24 Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.

25 Pierre Wolper. *Synthesis of Communicating Processes from Temporal-Logic Specifications*. PhD thesis, Stanford University, 1982.

26 Masafumi Yamashita and Tiko Kameda. Computing on an anonymous network. In *Proc. PODC*, pages 117–130, 1988.

Approximation Algorithms for Stochastic k-TSP*

Alina Ene[1], Viswanath Nagarajan[2], and Rishi Saket[3]

1 **Computer Science Department, Boston University, Boston, USA**
aene@bu.edu
2 **Industrial and Operations Engineering Department, University of Michigan, Ann Arbor, USA**
viswa@umich.edu
3 **IBM Research India, Bangalore. India**
rissaket@in.ibm.com

Abstract

This paper studies the stochastic variant of the classical k-TSP problem where rewards at the vertices are independent random variables which are instantiated upon the tour's visit. The objective is to minimize the expected length of a tour that collects reward at least k. The solution is a policy describing the tour which may (*adaptive*) or may not (*non-adaptive*) depend on the observed rewards. Our work presents an adaptive $O(\log k)$-approximation algorithm for Stochastic k-TSP, along with a non-adaptive $O(\log^2 k)$-approximation algorithm which also upper bounds the *adaptivity gap* by $O(\log^2 k)$. We also show that the adaptivity gap of Stochastic k-TSP is at least e, even in the special case of stochastic knapsack cover.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Stochastic TSP, algorithms, approximation, adaptivity gap

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.27

1 Introduction

In this paper, we consider the following stochastic variant of the classical k-TSP problem. The input consists of a metric (V, d) with depot $r \in V$. Each vertex $v \in V$ has an independent stochastic reward $R_v \in \mathbb{Z}_+$. All reward distributions are given as input and thus they are known upfront, but the actual reward instantiation R_v is only known when vertex v is visited. Given a target value k, the goal is to find an adaptive tour originating from r that collects a total reward at least k with the minimum expected length.[1] We assume that all rewards are supported on $\{0, 1, \ldots, k\}$.

Any feasible solution to this problem can be described by a decision tree where nodes correspond to vertices that are visited and branches correspond to observed reward instantiations. The size of such decision trees can be exponentially large. So we focus on obtaining solutions (aka policies) that implicitly specify decision trees. These solutions are called *adaptive* because the choice of the next vertex to visit depends on past random instantiations.

We will also consider the special class of *non-adaptive* solutions. Such a solution is described simply by an ordered list of vertices: the policy involves visiting vertices in the given order until the target of k is met (at which point the tour returns to r). These solutions are often preferred over adaptive solutions as they are easier to implement in

* Research done in part when the authors were at the IBM T. J. Watson Research Center.
[1] If the total instantiated reward $\sum_{v \in V} R_v$ happens to be less than k, the tour ends after visiting all vertices.

practice. However, the performance of non-adaptive solutions may be much worse, and so it is important to bound the *adaptivity gap* [15] which is the worst-case ratio between optimal adaptive and non-adaptive policies. This approach via non-adaptive policies has been very useful for a number of stochastic optimization problems, eg. [15, 19, 22, 5, 23, 24, 6, 25].

The Stochastic k-TSP problem captures several well-studied problems, including the classical k-TSP problem where the rewards are deterministic, and the *stochastic knapsack cover* problem where the metric (V, d) is a weighted star rooted at r: given n items with each item $i \in [n]$ having deterministic cost d_i and independent random reward $R_i \in \mathbb{Z}_+$, and a target k, find an adaptive policy that obtains total reward k at the minimum expected cost. The k-TSP problem (and the related spanning tree variant k-MST) have received considerable attention, leading to the development of a 2-approximation algorithm for the problem [18]. The stochastic knapsack cover problem was considered recently in [16], where an adaptive 2-approximation algorithm was obtained; however, no bounds on the adaptivity gap were known previously even for this special case.

Results and Techniques. In this paper, we give the first approximation algorithm for the general stochastic k-TSP, with an approximation ratio of $O(\log k)$. We also show that the adaptivity gap is $O(\log^2 k)$. The adaptivity gap is constructive: we give a non-adaptive algorithm that is an $O(\log^2 k)$-approximation to the optimal adaptive policy.

▶ **Theorem 1.** *There is an adaptive algorithm for stochastic k-TSP with approximation ratio $O(\log k)$. Moreover, the adaptivity gap is upper bounded by $O(\log^2 k)$.*

To motivate our approach, consider an algorithm for the classical k-TSP problem which has access to a hypothetical[2] subroutine exactly solving the deterministic *orienteering* problem: given a metric (V, d) on vertices with non-negative reward, a root vertex r, and a bound t compute a tour starting at r of length at most t which collects the maximum reward. The algorithm iterates over tour lengths of increasing powers of 2, solving for each length the deterministic orienteering problem, until the union of the computed tours yields reward k. It can be seen that this is an $O(1)$-approximation to k-TSP. Our algorithm adapts this iterative method to Stochastic k-TSP using the expected *truncated* rewards at each vertex. At the beginning of each iteration, the rewards distribution at each vertex is appropriately truncated w.r.t. the residual target (and set to zero if the vertex has been previously visited). However, in the stochastic setting, even if we used an exact orienteering algorithm, we need to construct roughly $(\log k)$ tours for each length (see Example 2 in Section 2.1). We show that using $O(\log k)$ many tours of each length suffices to obtain an $O(\log k)$-approximation algorithm for stochastic k-TSP. Moreover, this multiple-tour approach also allows for the use of an $O(1)$-approximation for orienteering, which is needed for a poly-time algorithm.

The high-level idea in the analysis is to show that the non-completion probability in the algorithm drops (roughly) by a constant factor in each iteration. This approach is similar to that for the classic min-latency TSP [10, 17]. Such an approach has also been used for stochastic optimization problems in [27, 28], but those results do not apply directly to metric-based costs that we need to handle here. Our main technical contribution is in proving the bound on non-completion probabilities in successive iterations (Lemma 3). This involves upper and lower bounding the expected "relative gain" which is the expected (truncated) increase in reward divided by the residual target. See Section 2 for details.

[2] We do not expect a poly-time exact algorithm for orienteering as the problem is APX-hard [9].

The non-adaptive algorithm is based on "simulating" the above adaptive algorithm. Corresponding to each orienteering instance solved in the adaptive algorithm we now solve $\log_2 k$ different instances based on different power-of-two values for the residual target. This results in a worse $O(\log^2 k)$ approximation ratio; this is relative to the optimal adaptive value and hence it bounds the adaptivity gap. We are not aware of a more direct approach for the non-adaptive problem, even if we compare to the optimal non-adaptive value.

We also show that the adaptivity gap is at least $e \approx 2.718$, which holds even in the special case of stochastic knapsack cover with a single random item.

▶ **Theorem 2.** *The adaptivity gap of* Stochastic k-TSP *(knapsack cover) is at least e.*

This result is obtained in an indirect manner. Using LP-duality we show that randomized online lower bounds for the *online bidding problem* [13] can be reduced to adaptivity gap lower bounds for stochastic knapsack-cover. These lower bound instances contain a *single* stochastic item. We can also show using a connection to the *incremental k-MST* problem [29], that the adaptivity gap for such instances is *exactly* e.

Other related work. Stochastic packing and covering problems have received considerable attention, leading to approximation guarantees and adaptivity gaps for several important problems, including knapsack [15, 8, 30, 16], packing and covering integer programs [14, 19], matching [12, 5, 1, 7, 2], matroid intersection [25, 3], submodular cover [20, 27, 21, 28], and orienteering [22, 24, 6]. Recently, [26] proved a constant adaptivity gap for submodular *maximization* problems under a very general class of constraints (including orienteering). This is however not directly applicable to stochastic k-TSP because we have (i) a minimization objective with strict covering requirement and (ii) general random variables as opposed to just binary in [26].

Sections 2 and 3 describe and analyze the adaptive and non-adaptive algorithms respectively for Stochastic k-TSP. Together they prove Theorem 1. The lower bound on the adaptivity gap (Theorem 2) is proved in Section 4.

2 Stochastic k-TSP Algorithm

Let us begin with an informal description of the adaptive algorithm. The algorithm relies on iteratively solving instances of the (deterministic) orienteering problem. In the orienteering problem, we are given a metric (V, d), depot r, profits at vertices and a length bound B; the goal is to find a tour originating from r of length at most B that maximizes the total profit. There is a $(2+\epsilon)$-approximation algorithm for orienteering [11] which we can use directly. The vertex-profits in the orienteering instance are chosen to be truncated expectations of the random rewards R_v where the truncation threshold is the current residual target. The length bounds are geometrically increasing with the phases of the algorithm. The algorithm terminates when the residual target reaches zero or all the vertices have been visited. However, in each phase the algorithm solves $\alpha \approx (\log k)$ many iterations each approximately solving the residual orienteering instance with the same length bound of that phase. This (roughly speaking) results in the $O(\log k)$ approximation ratio. The requirement of many iterations per phase turns out to be a somewhat subtle issue: we show in Section 2.1 that significantly reducing the number of repetitions results in a much worse approximation ratio.

The formal algorithm is given below. We assume (by scaling) that the minimum positive distance in the metric (V, d) is one. At any point in the algorithm, S denotes the set of currently visited vertices, σ denotes the reward instantiations of S, and $k(\sigma)$ is the total observed reward. The number of iterations of the inner for-loop is $\alpha := c \cdot H_k$, where $c \geq 1$ is

Algorithm 1 Algorithm Ad-kTSP

initialize $\sigma \leftarrow \emptyset$, $S \leftarrow \emptyset$ and $k(\sigma) = 0$.
for phase $i = 0, 1, \ldots$ **do**
  for $t = 1, \ldots, \alpha$ **do**
    define profits at vertices as follows

$$w_v := \begin{cases} \mathbb{E}[\min\{R_v, k - k(\sigma)\}] & \forall\, v \in V \setminus S \\ 0 & \forall\, v \in S \end{cases}$$

    using a ρ-approximation algorithm for the *orienteering* problem, compute a tour π originating from r of length at most 2^i with maximum total profit.
    traverse tour π and observe the actual rewards; augment S and σ accordingly.
    if $k(\sigma) \geq k$ or all vertices have been visited, the solution ends.
  end for
end for

a constant to be fixed later and $H_k \approx \ln k$ is the kth harmonic number. We will show that Ad-kTSP is an 8α-approximation algorithm for Stochastic k-TSP.

We view each iteration of the outer for loop as a *phase* and use i to index the phases. For any phase $i \geq 0$, we define the following quantities:

$$u_i := \Pr[\text{Ad-kTSP continues beyond phase } i]$$
$$u_i^* := \Pr\left[\text{optimal policy continues beyond distance } 2^i\right]$$

Since the minimum distance in the metric is one we have $u_0^* = 1$.

Overview of analysis

For analyzing the algorithm, observe that the expected cost of the obtained solution is roughly bounded by $\alpha \sum_{i \geq 0} u_i 2^{i+1}$ as the distance in phase i is roughly $\alpha \cdot 2^i$. To bound this we show that (*) $u_i - u_{i^*} \leq u_{i-1}/4$ which allows us to relate the expected cost of the algorithm with the expected optimal cost yielding an $O(\alpha)$ approximation. To show (*), the first step (Lemma 4) is proving the existence, in the ith phase with "state" σ, of an orienteering solution of length at most 2^i with profit at least a fraction $p_{i^*}(\sigma) := (1 - u_{i^*}|_\sigma)$ of the current residual target. Here $u_{i^*}|_\sigma$ corresponds to the probability u_{i^*} *conditioned* on the current state σ. Combined with a known inequality (Theorem 6) on the truncated sum of independent random variables, this is used to lower bound the expected *gain* (fraction of the residual target covered) of the adaptive algorithm in each phase by $\Omega(\alpha) \cdot (u_i - u_{i^*})$; see Claim 8. The expected gain on the other hand can easily be upper bounded by $H_k \cdot u_{i-1}$ where $H_k \approx \ln k$ is the kth harmonic number (Claim 7). Finally, setting $\alpha = \Theta(\log k)$ with a suitable constant finishes the proof.

Now to the formal details. The following lemma is the main component of the analysis.

▶ **Lemma 3.** *For any phase $i \geq 1$, we have $u_i \leq \frac{u_{i-1}}{4} + u_i^*$.*

Using Lemma 3, we can finish the analysis as follows. Let ALG denote the expected length of the tour constructed by the Ad-kTSP algorithm. Let OPT denote the expected length of the optimal adaptive tour. The total distance traveled by the Ad-kTSP algorithm in the first i phases is at most $\alpha \sum_{j=0}^{i} 2^j \leq 2^{i+1}\alpha$. Using this we obtain $\text{ALG} \leq 2\alpha \sum_{i \geq 1} 2^i u_i + 4\alpha$.

Also, $\text{OPT} \geq \frac{1}{2}\sum_{i\geq 1} 2^i u_i^* + u_0^* = \frac{1}{2}\sum_{i\geq 1} 2^i u_i^* + 1$. Letting $T := \sum_{i\geq 1} 2^i u_i$ and using Lemma 3, we obtain

$$T \;\leq\; \frac{1}{4}\sum_{i\geq 1} 2^i \cdot u_{i-1} + \sum_{i\geq 1} 2^i \cdot u_i^* \;\leq\; \frac{1}{2}\sum_{i\geq 1} 2^i \cdot u_i + \sum_{i\geq 1} 2^i \cdot u_i^* + \frac{1}{2} \;\leq\; \frac{1}{2}T + 2\cdot\text{OPT} - \frac{3}{2}.$$

It follows that $T \leq 4\cdot\text{OPT} - 3$ and $\text{ALG} \leq 8\alpha\cdot\text{OPT}$. Thus we obtain an 8α-approximation algorithm, which proves the first part of Theorem 1.

Now we turn to the proof of Lemma 3. We start by introducing some notation. Consider an arbitrary point in the execution of algorithm Ad-kTSP, and let $\langle S, \sigma, k(\sigma)\rangle$ be a triple in which S is the set of vertices visited so far, σ is the observed instantiation of S, and $k(\sigma)$ is the total reward observed in S. We refer to such a triple $\langle S, \sigma, k(\sigma)\rangle$ as the *state* of the algorithm.

▶ **Lemma 4.** *Consider a state $\langle S, \sigma, k(\sigma)\rangle$ of the* Ad-kTSP *algorithm. Consider the* Orienteering *instance in which each vertex in S is assigned a profit of zero and each vertex v not in S is assigned a profit of $\mathbb{E}[\min\{R_v, k - k(\sigma)\}]$. There is an orienteering tour from r of length at most 2^i with profit at least $(k - k(\sigma))\cdot p_i^*(\sigma)$, where*

$$p_i^*(\sigma) = \Pr\left[\text{ optimal policy completes before distance } 2^i \mid \sigma\right].$$

Proof. Consider the tree $\mathcal{T}^*$ representing the optimal policy. We *condition* on the instantiations σ on vertices S to obtain tree $\mathcal{T}^*(S,\sigma)$. Formally, we start with $\mathcal{T}^*(S,\sigma) = \mathcal{T}^*$ and apply the following transformation for each vertex $v \in S$ with instantiation σ_v: at each node ν in $\mathcal{T}^*(S,\sigma)$ corresponding to v, we remove all subtrees of ν except the subtree corresponding to instantiation σ_v; additionally, the probability of this edge (labeled σ_v) is set to one. Note that the probabilities at nodes corresponding to vertices $V \setminus S$ are unchanged, since rewards at different vertices are independent.

Now, *mark* those nodes in $\mathcal{T}^*(S,\sigma)$ that correspond to completion (i.e. reward at least k is collected) within distance 2^i. Note that the probability of reaching a marked node is exactly $p_i^*(\sigma)$. Finally, let $\mathcal{T}$ denote the subtree of $\mathcal{T}^*(S,\sigma)$ containing only those nodes that have a marked descendant. When tree $\mathcal{T}$ is traversed, the probability of reaching a leaf-node is $p_i^*(\sigma)$; with the remaining probability the traversal ends at some internal node. Clearly, every leaf-node in $\mathcal{T}$ is marked and hence corresponds to an r-tour of length at most 2^i along with instantiated rewards that sum to at least k. Define modified rewards at nodes of $\mathcal{T}$ as follows:

$$\hat{R}_v = \begin{cases} \min\{R_v, k - k(\sigma)\} & \text{if } v \notin S \\ 0 & \text{if } v \in S \end{cases}$$

Notice that the profits in the deterministic Orienteering instance are precisely $w_v = \mathbb{E}[\hat{R}_v]$. Observe that the sum of modified rewards at each leaf of $\mathcal{T}$ is at least $k - k(\sigma)$. Thus the expected $\hat{R}$-reward obtained in $\mathcal{T}$ is $\hat{E} \geq (k - k(\sigma))\cdot p_i^*(\sigma)$. Also,

$$\hat{E} = \sum_{\nu\in\mathcal{T}} \Pr[\text{reach } \nu]\cdot\mathbb{E}\left[\hat{R}_\nu \mid \text{reach } \nu\right] = \sum_{\nu\in\mathcal{T}} \Pr[\text{reach } \nu]\cdot\mathbb{E}\left[\hat{R}_\nu\right] = \sum_{\nu\in\mathcal{T}} \Pr[\text{end at } \nu]\cdot\left(\sum_{\mu\preceq\nu} w_\mu\right).$$

Above, for a node $\nu \in \mathcal{T}$, we use $\hat{R}_\nu$ and w_ν to denote the respective variables for the vertex (in V) corresponding to ν. Also, the notation $\mu \preceq \nu$ refers to node μ being an ancestor of node ν in $\mathcal{T}$. The first equality is by definition of $\hat{E}$, the second uses the fact that $\{\hat{R}_v : v \in V\}$ are independent, and the last is an interchange of summation. By averaging, there is some

node $\nu' \in \mathcal{T}$ with $\sum_{\mu \prec \nu'} w_\mu \geq \hat{E} \geq (k - k(\sigma)) \cdot p_i^*(\sigma)$. We can assume that ν' is a leaf node (otherwise we can reset ν' to be any leaf node of $\mathcal{T}$ below ν'). So the r-tour corresponding to this node ν' has length at most 2^i and is feasible for the deterministic ORIENTEERING instance. Moreover, it has profit at least $(k - k(\sigma)) \cdot p_i^*(\sigma)$ as claimed. ◀

▶ **Lemma 5.** *Consider a state $\langle S, \sigma, k(\sigma)\rangle$ of the* AD-KTSP *algorithm with $k(\sigma) < k$. Consider the* ORIENTEERING *instance with profits $\{w_v : v \in V\}$ where $w_v = 0$ for $v \in S$ and $w_v = \mathbb{E}\left[\min\{R_v, k - k(\sigma)\}\right]$ for $v \notin S$. Let π be any ρ-approximate orienteering tour consisting of vertices $V(\pi)$. Then,*

$$\mathbb{E}\left[\min\left\{\sum_{v \in V(\pi)\setminus S} R_v,\, k - k(\sigma)\right\}\right] \quad \geq \quad \frac{1}{\rho}\cdot\left(1 - \frac{1}{e}\right)\cdot(k - k(\sigma))\cdot p_i^*(\sigma).$$

Proof. For each $v \in V(\pi) \setminus S$ define $X_v := \min\left\{\frac{R_v}{k-k(\sigma)}, 1\right\}$. Note that X_vs are independent $[0,1]$ random variables, and $\mathbb{E}[X_v] = \frac{w_v}{k-k(\sigma)}$. Let $X := \sum_{v \in V(\pi)\setminus S} X_v$ and $Y = \min(X, 1)$. So the profit obtained by solution π to the deterministic orienteering instance is $(k - k(\sigma)) \cdot \mathbb{E}[X]$. Using Lemma 4 and the fact that π is a ρ-approximate solution, we obtain $\mathbb{E}[X] \geq \frac{1}{\rho} \cdot p_i^*$. We can now complete the proof of the lemma by applying Theorem 6 below. ◀

▶ **Theorem 6** ([4])**.** *Given a set $\{X_v\}$ of independent $[0,1]$ random variables with $X = \sum X_v$ and $Y = \min(X, 1)$, we have $\mathbb{E}[Y] \geq (1 - 1/e) \cdot \min\{\mathbb{E}[X], 1\}$.*

Now we are ready to prove Lemma 3.

Proof of Lemma 3. Consider any phase $i \geq 1$ and let $\langle S, \sigma, k(\sigma)\rangle$ be the state of the AD-KTSP algorithm at the start of some iteration of the inner loop. Let π be the orienteering tour that the algorithm visits next. Define

$$\text{gain}(\langle S, \sigma\rangle) := \frac{\mathbb{E}\left[\min\left\{\sum_{v \in V(\pi)\setminus S} R_v,\, k - k(\sigma)\right\}\right]}{k - k(\sigma)} \quad \text{if } k(\sigma) < k,$$

and $\text{gain}(\langle S, \sigma\rangle) := 0$ if $k(\sigma) \geq k$. The quantity $\text{gain}(\langle S, \sigma\rangle)$ measures the expected fraction of the residual target that we cover after visiting π.

Let $I(\sigma) = [k(\sigma) \geq k]$ be an indicator variable that is equal to one if $k(\sigma) \geq k$ and it is equal to zero otherwise. We have:

$$\text{gain}(\langle S, \sigma\rangle) \quad \geq \quad \frac{1}{\rho}\cdot\left(1 - \frac{1}{e}\right)\cdot\left(p_i^*(\sigma) - I(\sigma)\right). \tag{1}$$

To see this, note that if $I(\sigma)$ is one, the gain is zero and the inequality above holds since $p_i^*(\sigma) \leq 1$; and if $I(\sigma)$ is zero, this inequality follows from Lemma 5.

Recall that there are α iterations of the inner loop in phase i, and we use $t \in [\alpha]$ to index the iterations. For each iteration t (of phase i), let S_t be the random variable denoting the vertices visited until iteration t, and let σ_t denote the reward instantiations of S_t. Define:

$$G_t \quad := \quad \mathop{\mathbb{E}}_{\langle S_t, \sigma_t\rangle}[\text{gain}(\langle S_t, \sigma_t\rangle)]$$

Let $\Delta = \sum_{t=1}^{\alpha} G_t$ denote the total gain in phase i. The proof relies on upper and lower bounding Δ, which is done in the next two claims.

▶ **Claim 7.** *We have $\Delta \leq H_k \cdot u_{i-1}$.*

Proof. Let $\Delta(q)$ denote the value of Δ conditioned on the instantiations q of all random variables. If Ad-kTSP (conditioned on q) finishes before phase i then gain in each iteration is zero and $\Delta(q) = 0$.

In the following, we assume that Ad-kTSP (conditioned on q) reaches phase i. Let $L \leq k$ denote the residual target at the start of phase i, and $J_1, \ldots, J_\alpha \in \mathbb{Z}_+$ the incremental rewards obtained in each of the α iteration of phase i; recall that all rewards are integral. Then,

$$\Delta(q) \;\leq\; \sum_{t=1}^{\alpha} \frac{J_t}{L - J_1 - \cdots - J_{t-1}} \;\leq\; \sum_{t=1}^{L} \frac{1}{t} \;=\; H_L \;\leq\; H_k$$

The claim now follows since the algorithm reaches phase i only with probability u_{i-1}. ◀

▶ **Claim 8.** *We have* $\Delta \geq \frac{\alpha}{\rho} \cdot \left(1 - \frac{1}{e}\right) \cdot (u_i - u_i^*)$.

Proof. For any $t \in [\alpha]$, by Inequality (1), we have

$$G_t \;=\; \mathop{\mathbb{E}}_{\langle S_t, \sigma_t \rangle}[\text{gain}(\langle S_t, \sigma_t \rangle)] \;\geq\; \frac{1}{\rho} \cdot \left(1 - \frac{1}{e}\right) \cdot \mathop{\mathbb{E}}_{\sigma_t}[p_i^*(\sigma_t) - I(\sigma_t)]$$

Let $p(t)$ be the probability that Ad-kTSP finishes by iteration t of phase i. Since the probabilities $p(t)$ are non-decreasing in t, we have $p(t) \leq p(\alpha) = \Pr[\text{Ad-kTSP finishes by phase } i] = 1 - u_i$. For a fixed iteration t, the possible outcomes of $\langle S_t, \sigma_t \rangle$ correspond to a partition of the overall sample space. So we have $\mathbb{E}_{\sigma_t}[I(\sigma_t)] = p(t) \leq 1 - u_i$ and $\mathbb{E}_{\sigma_t}[p_i^*(\sigma_t)] = \Pr[\text{optimal policy completes within distance } 2^i] = 1 - u_i^*$. This completes the proof since $\Delta = \sum_{t=1}^{\alpha} G_t$. ◀

It follows from the two claims that

$$\frac{\alpha}{\rho} \cdot (1 - 1/e) \cdot (u_i - u_i^*) \;\leq\; \Delta \;\leq\; H_k \cdot u_{i-1}.$$

Setting $\alpha = 4\rho \frac{e}{e-1} \cdot H_k$ implies the lemma. ◀

We conclude this section by showing that our analysis of Ad-kTSP is essentially tight, and that it is necessary for α to be $\tilde{\Omega}(\log k)$.

2.1 Tightness of analysis of Ad-kTSP

Example 1. The analysis of Ad-kTSP is tight up to constant factors, even in the deterministic setting. Consider an instance of deterministic knapsack cover with $k = 2^\ell$ (for large integer ℓ) and $\ell(\ell+1)$ items as follows. For each $i \in \{0, 1, \ldots, \ell\}$ there is one item of cost 2^i and reward 2^i, and $\ell - 1$ items of cost 2^i and reward 1. Clearly the optimal cost is 2^ℓ. However the algorithm will select in each iteration $i = 0, 1, \ldots, \ell - 3$, all ℓ items of cost 2^i: note that the reward from these items is at most $\ell^2 + \sum_{i=0}^{\ell-3} 2^i \leq \ell^2 + 2^{\ell-2} < 2^\ell$. So the algorithm's cost is at least $\ell \cdot 2^{\ell-3}$.

Example 2. A natural variant of Ad-kTSP is to perform the inner iterations (for each phase $i = 0, 1, \ldots$) a constant number of times instead of $\Theta(\log k)$. Next, we show an example where such variants perform poorly even with access to a hypothetical subroutine solving the deterministic orienteering problem exactly. It is worth noting that in the deterministic case, such an approach (using an exact orienteering algorithm *once* for each phase i) suffices to obtain an $O(1)$-approximation for k-TSP. However, if we used an $O(1)$-approximation for orienteering, then such an approach performs poorly even for the deterministic k-TSP.

Consider the variant of Ad-kTSP that performs $1 \le h = o\left(\frac{\log k}{\log\log k}\right)$ iterations in each phase i. Choose $t \in \mathbb{Z}_+$ and set $\delta = 1/(ht)$ and $k = (ht)^{2ht} \in \mathbb{Z}_+$. Note that $ht = \Theta(\log k/\log\log k)$. Define a star metric centered at the depot r, with $ht+1$ leaves $\{u_{ij} : 0 \le i \le t-1, 0 \le j \le h-1\} \bigcup \{w\}$. The distances are $d(r,u_{ij}) = 2^i$ for all $i \in [t]$ and $j \in [h]$; and $d(r,w) = 1$. Each u_{ij} contains three co-located items: one of deterministic reward $(1-\delta)\delta^{hi+j} \cdot k$, and two having reward $\delta^{hi+j} \cdot k$ with probability δ (and zero otherwise). Finally w contains a deterministic item of reward k. By the choice of parameters, all rewards are integer valued. The optimal cost is 2, just visiting vertex w.

Now consider the execution of the modified Ad-kTSP algorithm. The probability that all the random items in the u_{ij}-vertices have zero reward is $(1-\delta)^{2ht} \ge \Omega(1)$. Conditioned on this event, it can be seen inductively that in the j^{th} iteration of phase i (for all i and j),

- the total observed reward until this point is $k(1-\delta)\sum_{\ell=0}^{hi+j-1}\delta^\ell = k\left(1-\delta^{hi+j}\right)$.
- the algorithm's tour (and optimal solution to the orienteering instance) involves visiting just vertex u_{ij} and choosing the three items in u_{ij}, for a total profit of $(1+\delta)\cdot\delta^{hi+j}\cdot k$.

Thus the expected cost of this algorithm is $\Omega(h \cdot 2^t)$, implying an approximation ratio $\exp\left(\frac{\log k}{h\cdot\log\log k}\right)$.

3 Non-Adaptive Algorithm

For our non-adaptive algorithm we show that the previous adaptive policy can be simulated in a non-adaptive manner, resulting in an $O(\log^2 k)$-approximate non-adaptive algorithm. This also upper bounds the adaptivity gap by the same quantity, proving the second part of Theorem 1. The difference from the adaptive setting is that the non-adaptive algorithm does not know the reward accrued so far and thus tries many possible residual targets for the orienteering problem. Specifically, at each of the α iterations in a phase, we append $\ell \approx \log_2 k$ different solutions to the orienteering problem defined with residual targets $k/2^j, j = 0, 1, \ldots, \lfloor \log_2 k \rfloor$. The analysis is almost identical to that of the adaptive algorithm. As before, we set $\alpha = O(H_k)$, yielding a $O(\ell\alpha) = O(\log^2 k)$ approximation.

The formal algorithm NonAd-kTSP is given below. The non-adaptive tour τ is constructed iteratively; S denotes the set of vertices visited in the current tour.

The difference from Ad-kTSP is in the inner for loop (indexed by j); this handles the fact that (unlike Ad-kTSP) the non-adaptive algorithm does not know the reward accrued so far. We set $\alpha := c' \cdot H_k$, where c' is a constant to be fixed later. Let $\ell := 1 + \lfloor \log_2 k \rfloor$ the number of iterations of the innermost loop.

The analysis for NonAd-kTSP is almost identical to Ad-kTSP. We will show that this is an $(8\alpha\ell)$-approximation algorithm for Stochastic k-TSP. As before, each iteration of the outer for loop is called a *phase*, which is indexed by i. For any phase $i \ge 0$, define

$$\begin{aligned} u_i &:= \Pr\left[\text{NonAd-kTSP continues beyond phase } i\right] \\ u_i^* &:= \Pr\left[\text{optimal adaptive policy continues beyond distance } 2^i\right] \end{aligned}$$

Just as in Lemma 3 we will show:

$$u_i \le \frac{u_{i-1}}{4} + u_i^*, \qquad \forall i \ge 1. \tag{2}$$

Since the total distance traveled by NonAd-kTSP in the first i phases is at most $\ell\alpha\sum_{h=0}^{i} 2^h \le \ell\alpha\, 2^{i+1}$, it follows (as for Ad-kTSP) that the expected length ALG of NonAd-kTSP is at most $8\ell\alpha \cdot \mathrm{OPT}$.

Algorithm 2 Algorithm NonAd-kTSP

initialize $\tau, S \leftarrow \emptyset$.
for phase $i = 0, 1, \ldots$ **do**
  for $t = 1, \ldots, \alpha$ **do**
    for $j = 0, 1, \ldots, \lfloor \log_2 k \rfloor$ **do**
      define profits as follows

$$w_v := \begin{cases} \mathbb{E}[\min\{R_v, k/2^j\}] & \text{for } v \in V \setminus S \\ 0 & \text{for } v \in S \end{cases}$$

      using a ρ-approximation algorithm for the *orienteering* problem, compute a tour π originating from r of length at most 2^i with maximum total profit.
      append tour π to τ, i.e. $\tau \leftarrow \tau \circ \pi$.
    end for
  end for
end for

We now prove (2). Consider a fixed phase $i \geq 1$ and one of the α iterations of the second for-loop (indexed by t). Let S denote the set of vertices visited before the start of this iteration $\langle i, t \rangle$. Let σ denote the reward instantiations at S, and $k(\sigma)$ the total reward in σ. Note that σ is only used in the analysis and not in the algorithm. Let $V(i,t) \subseteq V \setminus S$ be the new vertices visited in iteration $\langle i, t \rangle$; these come from ℓ different subtours corresponding to the inner for-loop. We will show (analogous to Lemma 5) that:

$$\mathbb{E}\left[\min\left\{\sum_{v \in V(i,t)} R_v,\, k - k(\sigma)\right\}\right] \quad \geq \quad \frac{1}{2\rho} \cdot \left(1 - \frac{1}{e}\right) \cdot (k - k(\sigma)) \cdot p_i^*(\sigma). \tag{3}$$

Above, $p_i^*(\sigma) = \Pr\left[\text{ optimal adaptive policy completes before distance } 2^i \mid \sigma\right]$. Exactly as in the proof of Lemma 3, this would imply (2) when we set $\alpha = 8\rho\frac{e}{e-1} \cdot H_k$.

It remains to prove (3). Let $g \in \{0, 1, \ldots, \ell\}$ denote the index such that $\frac{k}{2^g} \leq k - k(\sigma) < \frac{k}{2^{g-1}}$ and let $c := k/2^g$. An identical proof to Lemma 4 implies:

▶ **Lemma 9.** *Consider the* Orienteering *instance with profits* $s_v := \mathbb{E}[\min\{R_v, c\}]$ *for* $v \in V \setminus S$ *and* $s_v := 0$ *otherwise. There is an orienteering tour of length at most* 2^i *with profit at least* $c \cdot p_i^*(\sigma)$.

Let $T \subseteq V \setminus S$ denote the vertices added in iteration $\langle i, t \rangle$ corresponding to inner loop iterations $j \in \{0, 1, \ldots, g-1\}$. Note that in the inner loop iteration $j = g$, we have profits $w_v = \mathbb{E}[\min\{R_v, c\}]$ for $v \in V \setminus (S \cup T)$ and $w_v = 0$ otherwise. Lemma 9 now implies that the optimal profit of the orienteering instance in iteration $j = g$ is at least $c \cdot p_i^*(\sigma) - \sum_{u \in T} \mathbb{E}[\min\{R_u, c\}]$. Let T' denote the vertices added in the inner-loop iteration $j = g$. Since we obtain a ρ-approximate solution, it follows that the additional profit obtained $\sum_{v \in T'} w_v \geq \frac{1}{\rho}\left(c \cdot p_i^*(\sigma) - \sum_{u \in T} \mathbb{E}[\min\{R_u, c\}]\right)$. So,

$$\sum_{v \in V(i,t)} \mathbb{E}[\min\{R_v, c\}] \quad \geq \quad \sum_{v \in T \cup T'} \mathbb{E}[\min\{R_v, c\}] \quad \geq \quad \frac{c}{\rho} \cdot p_i^*(\sigma).$$

Exactly as in Lemma 5 we now obtain:

$$\mathbb{E}\left[\min\left\{\sum_{v\in V(\pi)\setminus S} R_v,\, c\right\}\right] \quad \geq \quad \frac{1}{\rho}\cdot\left(1-\frac{1}{e}\right)\cdot c\cdot p_i^*(\sigma).$$

And using $c \leq k - k(\sigma) < 2c$, we obtain (3).

In the following few paragraphs we give example showing that our analysis of NonAd-kTSP is tight, and that it is necessary for h to be $\tilde{\Omega}(\log k)$.

3.1 Tightness of analysis of NonAd-kTSP

Example 3. The analysis of NonAd-kTSP is tight up to constant factors, even in the deterministic setting. This example is similar to that for Ad-kTSP. Consider an instance of deterministic knapsack cover with $k = 2^\ell$ and $\ell^2(\ell+1)$ items as follows. For each $i \in \{0, 1, \ldots, \ell\}$ there is one item of cost 2^i and reward 2^i, and $\ell^2 - 1$ items of cost 2^i and reward 1. Clearly the optimal cost is 2^ℓ. However algorithm NonAd-kTSP will select in each iteration $i = 0, 1, \ldots, \ell-3$, all ℓ^2 items of cost 2^i: note that the reward from these items is at most $\ell^3 + \sum_{i=0}^{\ell-3} 2^i \leq \ell^3 + 2^{\ell-2} < 2^\ell$. So the algorithm's cost is at least $\ell^2 \cdot 2^{\ell-3}$, implying an approximation ratio of $\Omega(\log^2 k)$.

Example 4. One might also consider the variant of NonAd-kTSP where the second for-loop (indexed by t) is performed $1 \leq h \ll \log k$ times instead of $\Theta(\log k)$. The following example shows that such variants have a large approximation ratio. Consider an instance of stochastic knapsack cover with $k = 2^\ell$. There are an infinite number of items, each of cost one and reward k with probability $\frac{2}{3\ell}$ (the reward is otherwise zero). For each $i \in \{0, 1, \ldots, \ell/h\}$ and $j \in \{1, \ldots, \ell\}$ there are h items of cost 2^i and reward $k/2^j$ with probability $\frac{2}{3\ell}$ (the reward is otherwise zero). The optimal (adaptive and non-adaptive) policy considers only the cost one items with $\{0, k\}$ reward, and has optimal cost $O(\ell)$.

Algorithm NonAd-kTSP chooses in each iteration $i \in \{0, 1, \ldots, \ell/h\}$ (of the first for-loop), iteration $t \in \{1, \cdots, h\}$ (of the second for-loop) and iteration $j \in \{0, 1, \ldots, \ell\}$ (of the third for-loop) one of the items of cost 2^i with $\{0, k/2^j\}$ reward. (This is an optimal choice for the orienteering instance as rewards are capped at $k/2^j$ and the length bound is 2^i.) These items have total cost $\Theta(\ell \cdot 2^{\ell/h})$. For each $j \in \{0, 1, \ldots, \ell\}$ there are $\frac{\ell}{h} \cdot h = \ell$ items having reward $k/2^j$ with probability $\frac{2}{3\ell}$. It can be seen that the total reward from these items is less than k with constant probability. So the expected cost of NonAd-kTSP is $\Omega(\ell \cdot 2^{\ell/h})$, implying an approximation ratio $\Omega(2^{\ell/h})$.

4 Lower Bound on Adaptivity Gap

We show that the adaptivity gap of Stochastic k-TSP is at least $e \approx 2.718$ (Theorem 2). This holds even in the special case of the *stochastic covering knapsack* problem with a single random item. For such instances, we can even prove that the adaptivity gap is exactly e.

Here is an overview of the proof. We start by embedding an online bidding problem into a stochastic knapsack-cover instance such that non-adaptive (adaptive) policies for the latter correspond to online (offline) bidding strategies for the former. A result of Chrobak et al. [13] shows that no online bidding strategy is better than e-competitive. To complete the analysis, LP duality is used to show that the worst possible adaptivity gap for the stochastic knapsack-cover instance equals the best possible competitiveness of any online bidding strategy.

The gap example is based on the following problem studied in [13].

▶ **Definition 10** (Online Bidding Problem)**.** Given input $n \in \mathbb{Z}_+$, an algorithm outputs a randomized sequence $b_1, b_2, \ldots, b_\ell$ of bids from the set $[n] := \{1, 2, \ldots, n\}$. The algorithm's cost under (an unknown) threshold $T \in [n]$ equals the (expected) sum of its bid values until it bids a value at least T. The algorithm is β-competitive if its cost under threshold T is at most $\beta \cdot T$, for all $T \in [n]$.

▶ **Theorem 11** ([13])**.** *There is no randomized algorithm for online bidding that is less than e-competitive.*

Without loss of generality, any bid sequence must be increasing; let Γ denote the set of increasing sequences on $[n]$. For any $I \in \Gamma$ and $T \in [n]$, let $C(I,T)$ denote the cost of sequence I under threshold T. In terms of this notation, Theorem 11 is equivalent to:

$$\min_{\pi\,:\,\text{distribution}(\Gamma)} \max_{T\in[n]} \frac{\mathbb{E}_{I\leftarrow\pi}[C(I,T)]}{T} \quad \geq \quad e. \tag{4}$$

We now define the stochastic covering knapsack instance. The target $k := 2^{n+1}$. There is one random item r of zero cost having reward $k - 2^i$ with probability p_i (for all $i \in [n]$). There are n deterministic items $\{u_i\}_{i=1}^n$ where u_i has cost i and reward 2^i. We will show that there exist probabilities p_is so that the adaptivity gap is e.

It is clear that an optimal policy (adaptive or non-adaptive) will first choose item r, since it has zero cost. Moreover, an optimal adaptive policy will next choose item u_i exactly when r is observed to have reward $k - 2^i$. Hence the optimal adaptive cost is $\sum_{i=1}^n i \cdot p_i$.

Any non-adaptive policy is given by a sequence τ of the deterministic items; recall that item r is always chosen first. Moreover, due to the exponentially increasing rewards, we can assume that τ is an increasing sub-sequence of $\{u_i : 1 \leq i \leq n\}$. So there is a one-to-one correspondence between non-adaptive policies and Γ (in the online bidding instance). Note that the cost of non-adaptive solution $I \in \Gamma$ is exactly $\sum_{T=1}^n p_T \cdot C(I,T)$; if the random item r has size $k - 2^T$ then the policy will keep choosing items in I until it reaches an index at least T. Thus, the maximum adaptivity gap achieved by such an instance is:

$$\max_{p\,:\,\text{distribution}([n])} \min_{I\in\Gamma} \frac{\mathbb{E}_{T\leftarrow p}[C(I,T)]}{\mathbb{E}_{T\leftarrow p}[T]}. \tag{5}$$

The next lemma relates the quantities in (4) and (5). Below, for any set S, $\mathcal{D}(S)$ denotes the collection of probability distributions on S.

▶ **Lemma 12.** *For any non-negative matrix $C(\Gamma, [n])$, we have:*

$$\max_{p\in\mathcal{D}([n])} \min_{I\in\Gamma} \frac{\mathbb{E}_{T\leftarrow p}[C(I,T)]}{\mathbb{E}_{T\leftarrow p}[T]} \quad = \quad \min_{\pi\in\mathcal{D}(\Gamma)} \max_{T\in[n]} \frac{\mathbb{E}_{I\leftarrow\pi}[C(I,T)]}{T}.$$

Proof. The second expression equals the LP:

$$\begin{aligned} \min \quad & \beta \\ \text{s.t.} \quad & T\cdot\beta - \textstyle\sum_{I\in\Gamma} C(I,T)\cdot\pi(I) \geq 0, \qquad \forall T \in [n], \\ & \textstyle\sum_{I\in\Gamma} \pi(I) \geq 1, \\ & \beta, \pi \geq 0. \end{aligned}$$

Taking the dual, we obtain:

$$\begin{aligned} \max \quad & \alpha \\ \text{s.t.} \quad & \alpha - \textstyle\sum_{T\in[n]} C(I,T)\cdot\sigma(T) \leq 0, \qquad \forall I \in \Gamma, \\ & \textstyle\sum_{T\in[n]} T\cdot\sigma(I) \leq 1, \\ & \alpha, \sigma \geq 0. \end{aligned}$$

Define functions $g, f : [n] \to \mathbb{R}_+$ where

$$f(p) := \min_{I \in \Gamma} \sum_{T=1}^{n} p_T \cdot C(I,T) \quad \text{and} \quad g(p) := \sum_{T=1}^{n} p_T \cdot T.$$

Note that when p corresponds to a probability distribution, $f(p) = \min_{I \in \Gamma} \mathbb{E}_{T \leftarrow p}[C(I,T)]$ and $g(p) = \mathbb{E}_{T \leftarrow p}[T]$. So the first expression in the lemma is just $\max_{p \in \mathcal{D}([n])} f(p)/g(p)$. The key observation is that f and g are homogeneous, i.e. $f(a \cdot p) = a \cdot f(p)$ and $g(a \cdot p) = a \cdot g(p)$ for any scalar $a \in \mathbb{R}_+$ and vector $p \in \mathbb{R}^n_+$. This implies that the first expression equals:

$$\max_{p \in \mathcal{D}([n])} \frac{f(p)}{g(p)} \quad = \quad \max_{p \in \mathbb{R}^n_+} \frac{f(p)}{g(p)} \quad = \quad \max_{p \in \mathbb{R}^n_+ \,:\, g(p) \le 1} f(p).$$

It is easy to check that this equals the dual LP above, which proves the lemma. ◀

Thus the above instance of stochastic knapsack cover has adaptivity gap at least e, which proves Theorem 2. It can also be shown that every instance of stochastic knapsack cover with a single stochastic item has adaptivity gap at most e: this uses a relation to the incremental k-MST problem [29].

▶ **Proposition 13.** *The adaptivity gap of any instance of stochastic knapsack cover with a single random item is at most e.*

Proof. Consider any such instance with item n having random reward $R \in \{0, 1, \cdots k\}$ and each item $i \in [n-1]$ having deterministic reward r_i. Let c_i denote the cost of each $i \in [n]$. As there is only one random item, any adaptive policy is of the following form (i) select a subset $T_1 \subseteq [n-1]$ of deterministic items, (ii) select random item n, (iii) depending on the value of reward R, select subset $T_2(R) \subseteq [n-1]$ of deterministic items.

Note that if the reward in T_1 is itself at least k then we do not even need to use the random item: so the optimal adaptive and non-adaptive policies are the same (both are T_1). On the other hand, if $r(T_1) < k$ then the random item will be selected with probability one: so we can assume that it is the first item to be selected, i.e. $T_1 = \emptyset$.

Now, consider the following *incremental k-MST* instance. There are n vertices with distances induced by a star with root n and leaves $[n-1]$ where the distance $d(n,i) = c_i$. There is zero reward at the root and reward r_i at each $i \in [n-1]$. The goal is to find a random sequence σ of edges so that the following ratio is minimized:

$$\max_{\ell \ge 0} \quad \frac{\mathbb{E}_\sigma[\text{cost of minimal prefix of } \sigma \text{ with reward at least } \ell]}{\text{optimal } \ell\text{-MST value}}.$$

As shown in [29], there is always a random sequence σ with the above ratio at most e.

In the stochastic knapsack-cover instance let $p_\ell = \Pr[R = \ell]$ for $\ell \in \{0, 1, \cdots k\}$. Then it is clear that the adaptive optimum is $AD = c_n + \sum_{\ell=0}^{k} p_{k-\ell} \cdot OPT(\ell)$ where $OPT(\ell)$ is the optimal ℓ-MST value. Note also that the sequence σ corresponds to a randomized non-adaptive solution: item n followed by the remaining $n-1$ items in the order that they first appear in σ. This has expected value $NA = c_n + \sum_{\ell=0}^{k} p_{k-\ell} \cdot \mathbb{E}_\sigma[\sigma(\ell)]$ where $\sigma(\ell)$ is the cost of the minimal prefix of σ with reward at least ℓ. So the adaptivity gap is at most:

$$\frac{NA}{AD} = \frac{c_n + \sum_{\ell=0}^{k} p_{k-\ell} \cdot \mathbb{E}_\sigma[\sigma(\ell)]}{c_n + \sum_{\ell=0}^{k} p_{k-\ell} \cdot OPT(\ell)} \le \max_\ell \frac{\mathbb{E}_\sigma[\sigma(\ell)]}{OPT(\ell)} \le e.$$

This also implies that the best deterministic non-adaptive policy costs at most $e \cdot AD$. ◀

5 Conclusion

The main (perhaps challenging) open question is to obtain a constant-factor approximation algorithm for Stochastic k-TSP in either the adaptive or non-adaptive setting. Another interesting question is the adaptivity gap, even in the special case of stochastic knapsack cover, where there is a wide gap between the lower bound of e and upper bound of $O(\log^2 k)$.

Acknowledgment. We thank Itai Ashlagi for initial discussions that lead to this problem definition.

References

1 M. Adamczyk. Improved analysis of the greedy algorithm for stochastic matching. *Information Processing Letters*, 111(15):731–737, 2011.

2 M. Adamczyk, F. Grandoni, and J. Mukherjee. Improved approximation algorithms for stochastic matching. In *Proc. Annual European Symposium on Algorithms*, pages 1–12, 2015.

3 M. Adamczyk, M. Sviridenko, and J. Ward. Submodular stochastic probing on matroids. *Math. Oper. Res.*, 41(3):1022–1038, 2016.

4 Y. Azar, A. Madry, T. Moscibroda, D. Panigrahi, and A. Srinivasan. Maximum bipartite flow in networks with adaptive channel width. *Theoretical Computer Science*, 412(24):2577–2587, 2011.

5 N. Bansal, A. Gupta, J. Li, J. Mestre, V. Nagarajan, and A. Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4):733–762, 2012.

6 N. Bansal and V. Nagarajan. On the adaptivity gap of stochastic orienteering. *Mathematical Programming*, 154(1-2):145–172, 2015.

7 A. Baveja, A. Chavan, A. Nikiforov, A. Srinivasan, and P. Xu. Improved bounds in stochastic matching and optimization. In *Proc. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 124–134, 2015.

8 A. Bhalgat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *Proc. Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1647–1665. Society for Industrial and Applied Mathematics, 2011.

9 A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal of Computing*, 37(2):653–670, 2007.

10 K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. Annual Symposium on Foundations of Computer Science*, pages 36–45, 2003.

11 C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3):23, 2012.

12 N. Chen, N. Immorlica, A. R Karlin, M. Mahdian, and A. Rudra. Approximating matches made in heaven. In *Proc. International Colloquium on Automata, Languages and Programming*, pages 266–278. Springer, 2009.

13 M. Chrobak, C. Kenyon, J. Noga, and N. E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008.

14 B. C Dean, M. X. Goemans, and J. Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proc. Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 395–404. Society for Industrial and Applied Mathematics, 2005.

15 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008. doi:10.1287/moor.1080.0330.

16 A. Deshpande, L. Hellerstein, and D. Kletenik. Approximation algorithms for stochastic submodular set cover with applications to boolean function evaluation and min-knapsack. *ACM Transactions on Algorithms*, 12(3):42, 2016.

17 J. Fakcharoenphol, C. Harrelson, and S. Rao. The k-traveling repairmen problem. *ACM Transactions on Algorithms*, 3(4), 2007.

18 N. Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In *Proc. ACM Symposium on the Theory of Computing*, pages 396–402, 2005.

19 M. X. Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *Proc. Latin American Symposium on Theoretical Informatics(LATIN)*, pages 532–543, 2006.

20 D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

21 N. Grammel, L. Hellerstein, D. Kletenik, and P. Lin. Scenario submodular cover. In *International Workshop on Approximation and Online Algorithms*, pages 116–128. Springer, 2016.

22 S. Guha and K. Munagala. Multi-armed bandits with metric switching costs. *Automata, Languages and Programming*, pages 496–507, 2009.

23 A. Gupta, R. Krishnaswamy, M. Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *Proc. Annual Symposium on Foundations of Computer Science*, pages 827–836, 2011.

24 A. Gupta, R. Krishnaswamy, V. Nagarajan, and R. Ravi. Running errands in time: Approximation algorithms for stochastic orienteering. *Math. Oper. Res.*, 40(1):56–79, 2015.

25 A. Gupta and V. Nagarajan. A stochastic probing problem with applications. In *Proc. Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 205–216, 2013.

26 A. Gupta, V. Nagarajan, and S. Singla. Adaptivity gaps for stochastic probing: Submodular and XOS functions. In *Proc. Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1688–1702, 2017.

27 Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016.

28 P. Kambadur, V. Nagarajan, and F. Navidi. Adaptive submodular ranking. In *Proc. Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 317–329, 2017.

29 G. Lin, C. Nagarajan, R. Rajaraman, and D. P. Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal of Computing*, 39(8):3633–3669, 2010.

30 W. Ma. Improvements and generalizations of stochastic knapsack and multi-armed bandit approximation algorithms. In *Proc. Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1154–1163. Society for Industrial and Applied Mathematics, 2014.

Synthesis in Distributed Environments*†

Bernd Finkbeiner[1] and Paul Gölz[2]

1 **Saarland University, Saarbrücken, Germany**
finkbeiner@react.uni-saarland.de
2 **Saarland University, Saarbrücken, Germany**
pgoelz@cs.cmu.edu

Abstract

Most approaches to the synthesis of reactive systems study the problem in terms of a two-player game with complete observation. In many applications, however, the system's environment consists of several distinct entities, and the system must actively communicate with these entities in order to obtain information available in the environment. In this paper, we model such environments as a team of players and keep track of the information known to each individual player. This allows us to synthesize programs that interact with a distributed environment and leverage multiple interacting sources of information.

The synthesis problem in distributed environments corresponds to solving a special class of Petri games, i.e., multi-player games played over Petri nets, where the net has a distinguished token representing the system and an arbitrary number of tokens representing the environment. While, in general, even the decidability of Petri games is an open question, we show that the synthesis problem in distributed environments can be solved in polynomial time for nets with up to two environment tokens. For an arbitrary but fixed number of three or more environment tokens, the problem is NP-complete. If the number of environment tokens grows with the size of the net, the problem is EXPTIME-complete.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases reactive synthesis, distributed information, causal memory, Petri nets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.28

1 Introduction

Automating the creation of programs is one of the most ambitious goals in computer science. Given a specification, a synthesis algorithm either generates a program that satisfies the specification or determines that no such program exists. The promise of synthesis is to let programmers work on a more abstract level and thus to fundamentally simplify the development of complex software.

Most current synthesis approaches (cf. [16, 5, 3, 15, 7]) are based on the game-theoretic approach, originally introduced by Büchi and Landweber [4], in which the synthesis problem is seen as a two-player game with complete observation, played between a *system* player and an *environment* player. The goal of the system player is to ensure that the specification is satisfied; the goal of the environment player is to ensure a violation. A winning strategy for the system player defines a control program that reads in the decisions of the environment as its inputs and produces the decisions of the system as its outputs.

* Supported by the European Research Council (ERC) Grant OSARES (No. 683300).
† Omitted proofs, notation for multisets and the algorithm mentioned in Theorems 6 and 9 can be found in the full version [11].

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 28; pp. 28:1–28:14

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

A fundamental limitation of the standard game-theoretic formulation is that the environment is a monolithic block. In many applications, however, the environment consists of several distinct entities, and the system must actively communicate with these entities in order to obtain information available in the environment. In this paper, we introduce the *synthesis problem in distributed environments*. As in the standard approach, we view the synthesis problem as a game between the system and the environment. However, rather than considering the environment as a single *player* in this game, we consider it as a *team* consisting of several players that may carry different information. Both the individual environment players and the system player can increase their knowledge by interacting with other players.

The problem is related to, but very different from, the *distributed synthesis* problem [18]. In distributed synthesis, it is the *system* that is partitioned into multiple players, corresponding to multiple processes. The key difficulty here is to coordinate the strategies of the system players. In the synthesis problem in distributed environments, it is instead the *environment* that consists of multiple entities. The key difficulty here is for the system player to synchronize with the right environment players at the right points in time.

We study the synthesis problem in distributed environments in the framework of *Petri games* [12]. The players of a Petri game are represented as the tokens of a Petri net, partitioned into the system and environment players. Synthesis in distributed environments corresponds to Petri games with a single system token and multiple environment tokens. We assume that the underlying Petri net is bounded, i.e., only a bounded number of players can be generated over the course of a game. For unbounded nets, Petri games are known to be undecidable [12].

The players of a Petri game advance asynchronously except for synchronous interactions, in which players exchange knowledge. We assume that, whenever multiple players interact, they exchange information both truthfully and maximally. This model of knowledge is called *causal memory*. In this paper, we restrict our synthesis to *safety* specifications, i.e., the system must prevent the global state from entering certain bad configurations.

We illustrate our setting with a small access control example. Suppose you would like to synthesize a lock controller for a safe that contains sensitive business information. Corporate policy mandates that the safe may only be jointly opened by two employees and that both must previously have confirmed their identity with a corresponding authentication authority. The environment of the lock controller thus consists of four independent players: the employees e^1 and e^2 and their authenticators a^1 and a^2. These entities interact with each other (when a^1 authenticates e^1 or a^2 authenticates e^2) and with the system player (when e^1 or e^2 request the safe to open). Since there is no direct interaction between the lock controller and the authenticators, the knowledge about the authentication must be provided to the lock controller by the employees.[1]

Figure 1 shows how our access control scenario can be modeled as a Petri game. Players are represented by tokens (dots) that move between places (circles) using transitions (squares). The system player, who only moves between places marked in gray, starts in a place indicating that the safe is closed. The game allows her to consult with any employee and remain in her position, or to move to the place s_{open} to open the safe. The first employee starts in e^1 and can either directly move to e^1_{attempt} or can synchronize with her authenticator to move there. In the latter case, the authenticator simultaneously moves to a^1_{auth}, where

[1] In Petri games, all players are truthful. Think of the tokens as carriers of information, e.g., a cryptographically secured smart card carried by the employee.

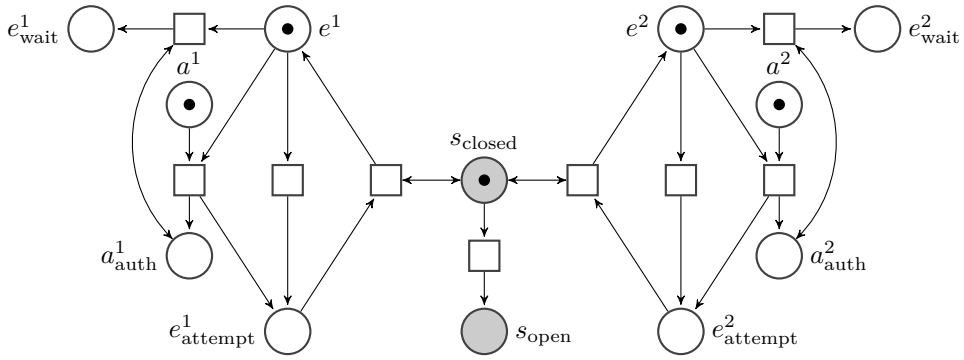


Figure 1 Petri game of the access control example. If the system player lies in s_{open} while there is still a player in a^1 or a^2, the system immediately loses the game.

she cannot authenticate e^1 a second time. When the employee is in e^1_{attempt}, the system player can choose to synchronize with her, moving the employee back to e^1 and exchanging all knowledge between the players. In particular, the system player learns whether the employee was authenticated. Afterwards, the employee can attempt to open the safe again, for example to make up for not being authenticated the last time. If the employee has already authenticated, she can alternatively move to e^1_{wait} and remain there. This possibility forces the locking mechanism to stop waiting for communication once it knows enough and to unlock the safe instead. The second employee is modeled symmetrically. To prevent the system from unlocking prematurely, we declare that all situations in which the safe is open but in which one authenticator has not moved yet as losing for the system.

A winning strategy for this game, as found by our synthesis algorithm, would be to allow communication with e^1 and nothing else until (possibly never) the system learns that the employee has authenticated. Then, it allows communication with e^2 until the same is true for the second employee. Finally, it opens the safe.

Related work. Synthesis in distributed environments is related to planning under partial observation [19, pp. 138–146] in that our strategies also combine information gathering and action. However, the classical partial-information setting does not capture the knowledge of different actors. With causal memory, a player's knowledge naturally refers to past observations and to the knowledge of other players. Synthesis in distributed environments can be expressed as a control problem [14, 17] for Zielonka's asynchronous automata [21]. Because this model is very expressive, all known decidability results assume strong restrictions on the communication architecture. Since our environment players are allowed to freely interact with each other and with the system, we cannot apply these results. Petri games were introduced in [12] and there is growing tool support for solving Petri games [10, 9]. The decidability of general Petri games is an open question. The only previously known decision procedure is restricted to the case of a single environment token [12]. In this paper, we solve the complementary case, where the number of environment tokens is unbounded (but there is only one system token). There is also a semi-algorithm for solving Petri games [8]. This approach finds finitely representable winning strategies, but does not terminate if no winning strategy exists.

Contributions. Our main technical contribution is an EXPTIME algorithm for deciding bounded Petri games with one system player and an arbitrary number of environment players.

Previously, the synthesis problem for Petri games with more than one environment player was open. We provide a matching lower bound to show that our algorithm is asymptotically optimal. If the number of environment players is kept constant, we show that the problem can be solved in polynomial time for up to two environment players whereas it is NP-complete for three or more environment players. The following table sums up the complexity of deciding k-bounded Petri games with one system player and e environment players, for any $k \geq 1$:

$e \leq 2$	P
$e \geq 3$	NP-complete
e grows with net	EXPTIME-complete

2 Petri nets

We recall notions from the theory of Petri nets as used in [12]. A tuple $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ is called a *Petri net* if it satisfies the following conditions:

- The set of *places* $\mathcal{P}$ and the set of *transitions* $\mathcal{T}$ are disjoint;
- The *flow relation* $\mathcal{F}$ is a multiset over $(\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$, i.e., $\mathcal{N}$ is a directed, bipartite multigraph with nodes $\mathcal{P} \cup \mathcal{T}$ and edges given by $\mathcal{F}$. We use the term *nodes* to refer to places and transitions simultaneously. For nodes x, y, we write $x \mathcal{F} y$ to denote $(x, y) \in \mathcal{F}$;
- The *initial marking* In is a finite multiset over $\mathcal{P}$;
- We require finite synchronization and nonempty pre- and postconditions: For a node x, define the *precondition* as a multiset $pre(x)$ such that $pre(x)(y) = \mathcal{F}(y, x)$ for all nodes y and similarly define the *postcondition* by $post(x)(y) = \mathcal{F}(x, y)$. Then, all transitions t must satisfy $0 < |pre(t)| < \infty$ and $0 < |post(t)| < \infty$.

A net is called *finite* if it contains finitely many nodes.

By convention, the components of a net $\mathcal{N}$ are named $\mathcal{P}$, $\mathcal{T}$, $\mathcal{F}$ and In, and similarly for nets named $\mathcal{N}_1$, $\mathcal{N}^\sigma$, $\mathcal{N}^U$, etc. We graphically specify Petri nets as multigraphs, where places are represented by circles, transitions by squares and the flow relation by arrows. In addition, the number of dots in a place reflects the multiplicity of this place in the initial marking. Apart from the gray color of certain places, Fig. 1 shows a Petri net with named places.

A *marking* $\mathcal{M}$ of $\mathcal{N}$ is a finite multiset over $\mathcal{P}$. We think of the Petri net as a board on which a finite number of *tokens* moves between places by using transitions. A marking then represents a certain configuration by listing the current number of tokens on every place. We can move from one marking to another by firing a transition t, i.e., by removing tokens in $pre(t)$ and putting tokens into $post(t)$ instead. If the total number of tokens changes in this process, we think of such transitions as generating or consuming tokens. We say that t is *enabled* in a marking $\mathcal{M}$ if $pre(t) \subseteq \mathcal{M}$. If this is the case, we can obtain a new marking $\mathcal{M}' := \mathcal{M} - pre(t) + post(t)$ by *firing* t, and we write $\mathcal{M} \,|t\rangle\, \mathcal{M}'$ to denote that $\mathcal{M}'$ can be constructed from $\mathcal{M}$ and t in this way. A marking is said to be *reachable* if it can be reached from the initial marking by firing a finite sequence of transitions. We generalize preconditions and postconditions to sets S of nodes by defining $pre(S) := \biguplus_{x \in S} pre(x)$ and analogously for $post(S)$. A Petri net is *k-bounded* for a natural number $k \geq 1$ if, for all reachable markings $\mathcal{M}$ and places p, $\mathcal{M}(p) \leq k$ holds. We call a net *bounded* if it is k-bounded for some k.

We are mainly interested in Petri nets as a model for the causal dependencies between events. These dependencies are made explicit in *occurrence nets*, certain acyclic nets in which each place has a unique causal history. Before giving their definition, we introduce notation to capture different kinds of causal relationships between nodes. We denote the transitive closure of the support of $\mathcal{F}$ by $<$ and its reflexive and transitive closure by $\leq$.

We call x and y *causally related* if $x \leq y$ or $y \leq x$. The *causal past* of a node x is the set $past(x) \coloneqq \{y \in \mathcal{P} \cup \mathcal{T} \mid y \leq x\}$. We extend this notion to sets S of nodes by setting $past(S) \coloneqq \bigcup_{x \in S} past(x)$. Apart from being causally related, two nodes x, y might also be mutually exclusive, i.e., they might be the result of alternative, nondeterministic choices. We say that x and y are *in conflict*, for short $x \sharp y$, if there exists a place p, $p \neq x, p \neq y$, such that x and y can be reached following the flow relation from p via different outgoing transitions. If x and y are neither causally related nor in conflict, we call them *concurrent*.

An *occurrence net* is a net $\mathcal{N}$ that satisfies all of the following conditions: the pre- and postconditions of transitions are sets, not general multisets; each place has at most one incoming transition; the initial marking is the set $\{p \in \mathcal{P} \mid pre(p) = \emptyset\}$; the inverse flow relation $\mathcal{F}^{-1}$ is well-founded, i.e., if we start from any node and follow the flow relation backwards, we eventually reach a place in the initial marking; no transition is in conflict with itself. Occurrence nets are 1-bounded, i.e., their reachable markings are sets.

We call a maximal set $\mathcal{C}$ of pairwise concurrent places in an occurrence net a *cut*. The *finite* cuts of an occurrence net are exactly its reachable markings [11, App. B.1]. Since the occurrence nets that we will work with only have finite cuts, we can use the terms interchangeably [11, Corollary 17].

A *homomorphism* from a Petri net $\mathcal{N}_1$ to a Petri net $\mathcal{N}_2$ is a function $\lambda : \mathcal{P}_1 \cup \mathcal{T}_1 \to \mathcal{P}_2 \cup \mathcal{T}_2$ that only maps places to places and transitions to transitions such that, for all $t \in \mathcal{T}_1$, $\lambda[pre(t)] = pre(\lambda(t))$ and $\lambda[post(t)] = post(\lambda(t))$. λ is called *initial* if additionally $\lambda[In_1] = In_2$ holds.

An *initial branching process* β of a net $\mathcal{N}$ is a pair $(\mathcal{N}^U, \lambda)$ where $\mathcal{N}^U$ is an occurrence net and λ is an initial homomorphism from $\mathcal{N}^U$ to $\mathcal{N}$ such that $\forall t_1, t_2 \in \mathcal{T}^U.\ (pre(t_1) = pre(t_2) \wedge \lambda(t_1) = \lambda(t_2)) \to t_1 = t_2$. Conceptually, a branching process describes a subset of the possible behavior of a net as an occurrence net. If a place or a transition in the original net can be reached on different paths or with different knowledge, the branching process splits up this node. The homomorphism λ is used to label those multiple instances with the original node in $\mathcal{N}$. The additional condition means that the branching process may not split up a transition unnecessarily: For the same precondition, at most one instance of a certain transition can be present in the branching process.

3 Petri games

In a Petri game, we partition the places of a finite Petri net into two disjoint subsets: the *system places* $\mathcal{P}_S$ (represented in gray) and the *environment places* $\mathcal{P}_E$ (represented in white). For convenience, we write $\mathcal{P}$ for the set of all places of the game $\mathcal{P}_S \cup \mathcal{P}_E$. A token on a system place represents a *system player*, a token on an environment place an *environment player*. Additionally, a Petri game also identifies a set of *bad markings* $\mathcal{B}$, which the system players need to avoid.[2] If the game reaches a marking $\mathcal{M}$ in $\mathcal{B}$, the environment wins; the system wins if this is never the case. Formally, a *Petri game* $\mathcal{G}$ is a tuple $(\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{B})$. We call $\mathcal{N}^{\mathcal{G}} \coloneqq (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ the *underlying net* of $\mathcal{G}$.

Transitions whose entire precondition belongs to the environment are called *purely environmental*. Otherwise, we call the transition a *system transition*.

[2] This is more general than in [12], where instead of avoiding a set of arbitrary markings, the system tries to avoid all markings that have a nonempty intersection with a set of bad places. [8] also uses arbitrary sets of bad markings. Since the hardness proofs in Theorems 7 and 8 only use bad markings of this shape, this generalization does not increase the computational hardness of our setting. In our complexity analyses in Theorems 6, 8 and 9, we do not commit to a specific input encoding of bad markings such that our results remain valid if a set of bad places is given instead.

Since Petri games aim to model the information flow in a system, a system player's decisions may only depend on information that she has witnessed herself or that she has obtained by communicating with other players. We thus describe strategies of the system as branching processes of the underlying net of the game, where the causal dependencies are made explicit. While the game is played on the underlying net, the strategy keeps track of the current state of the game as well as its causal history. Every reachable marking of the branching process corresponds to a reachable marking in the underlying net [11, Lemma 14]. However, the marking in the strategy might have less enabled transitions than the one in the underlying net, which means that the strategy can prevent certain transitions from firing. The game progresses by nondeterministically firing transitions that are allowed by the strategy. No matter which transitions are fired in which order, the system players need to ensure certain properties of the game. Because of this, it is sometimes useful to think of these choices as being made by an adversarial scheduler.

A winning, deadlock-avoiding *strategy* is an initial branching process $\beta_\sigma = (\mathcal{N}^\sigma, \lambda)$ of the underlying net of the game that satisfies the following four conditions:

justified refusal Let S be a set of pairwise concurrent places in $\mathcal{P}^\sigma$ and $\boldsymbol{t}$ be a transition *in the underlying net*, where $\lambda[S] = pre(\boldsymbol{t})$ but there is no $t \in \mathcal{T}^\sigma$ such that $\lambda(t) = \boldsymbol{t}$ and $pre(t) = S$. Then, there must be a place $s \in S \cap \mathcal{P}^\sigma_S$ such that $\boldsymbol{t} \notin \lambda(post(s))$.

safety For all $\mathcal{M} \in \mathcal{R}(\mathcal{N}^\sigma)$, $\lambda[\mathcal{M}] \notin \mathcal{B}$.

determinism For all $s \in \mathcal{P}^\sigma_S$ and all reachable markings $\mathcal{M}$ in $\mathcal{N}^\sigma$ that contain s, there is at most one transition $t \in post(s)$ that is enabled in $\mathcal{M}$.

deadlock avoidance For all $\mathcal{M} \in \mathcal{R}(\mathcal{N}^\sigma)$ we require that, if any transition of the underlying net is enabled in $\lambda[\mathcal{M}]$, then some transition in the strategy must be enabled in $\mathcal{M}$.

In the above conditions, we extended the notion of system places to the strategy by setting $\mathcal{P}^\sigma_S := \mathcal{P}^\sigma \cap \lambda^{-1}(\mathcal{P}_S)$. We similarly define the environment places of the strategy as $\mathcal{P}^\sigma_E := \mathcal{P}^\sigma \cap \lambda^{-1}(\mathcal{P}_E)$. To distinguish more clearly between nodes in the strategy and nodes in the underlying net, we always use bold variable names such as $\boldsymbol{p}$ or $\boldsymbol{t}$ for the latter.

Justified refusal means that a system player influences the course of the game by refusing to take part in certain transitions in her postcondition. Even if every place in $pre(\boldsymbol{t})$ contains a token for some $\boldsymbol{t} \in \mathcal{T}$, the transition can fire iff, for every place in $pre(\boldsymbol{t}) \cap \mathcal{P}_S$, the corresponding system player allows this transition. In particular, purely environmental transitions cannot be restricted by the strategy. More precisely, the condition refers to all possible preconditions S where a transition could have been added to the strategy, but was not. If no instance t of $\boldsymbol{t}$ with the right precondition exists so far, there must be a system place in S that refuses to take part in any instance of $\boldsymbol{t}$. Note that a system player can only refuse all transitions in the strategy with the label $\boldsymbol{t}$ or must allow all of them.

The safety objective requires that the game never reaches a bad marking. Determinism enforces that, from a system player's perspective, all sources of uncertainty are in the vicinity of an environment player. This does not prevent a system player from allowing multiple transitions, as long as these transitions are enabled in different markings.

Finally, we require the strategy to avoid deadlocks. Without this condition, a strategy might simply refuse to fire any system transition at all. In general, the system prefers to fire less transitions since they might potentially lead to bad markings and since allowing too many of them might cause nondeterminism. The criterion enforces that, whenever no purely environmental transition is enabled in a marking but some system transition is enabled, the strategy must allow one of them in order to keep the game going. This still allows the strategy to enter markings in which no transition is enabled at all. Similarly, a system player may refuse all transitions in her postcondition as long as she knows that the game will always allow another player to move.

$$\mathcal{V}_0' = \{(\mathcal{M}, \top, \{\boldsymbol{s_{\mathcal{M}}}\}) \mid \mathcal{M} \in \mathcal{R}(\mathcal{N})\}$$
$$\mathcal{V}_1' = \{(\mathcal{M}, c, R) \mid \mathcal{M} \in \mathcal{R}(\mathcal{N}); c \subseteq \mathit{post}(\boldsymbol{s_{\mathcal{M}}}); \boldsymbol{s_{\mathcal{M}}} \in R \subseteq \mathcal{M}\}$$
$$\mathcal{I}' = (\mathit{In}, \top, \{\boldsymbol{s_{In}}\})$$
$$\mathcal{E}' = \{(\mathcal{M}, \top, \{\boldsymbol{s_{\mathcal{M}}}\}) \to (\mathcal{M}, c, \{\boldsymbol{s_{\mathcal{M}}}\}) \mid c \subseteq \mathit{post}(\boldsymbol{s_{\mathcal{M}}})\} \tag{E'1}$$
$$\cup \left\{ (\mathcal{M}, c, R) \to (\mathcal{M}', c, R') \;\middle|\; \begin{array}{l} \boldsymbol{t} \text{ purely environmental transition}; \mathcal{M} \,|\boldsymbol{t}\rangle\, \mathcal{M}'; \\ \boldsymbol{o} \in \mathit{post}(\boldsymbol{t}); R' = R - \mathit{pre}(\boldsymbol{t}) + \{\boldsymbol{o}\} \end{array} \right\} \tag{E'2}$$
$$\cup \left\{ (\mathcal{M}, c, R) \to (\mathcal{M}', \top, \{\boldsymbol{s_{\mathcal{M}'}}\}) \;\middle|\; \begin{array}{l} \boldsymbol{t} \text{ system transition}; \boldsymbol{t} \in c; \mathcal{M} \,|\boldsymbol{t}\rangle\, \mathcal{M}'; \\ R \subseteq \mathit{pre}(\boldsymbol{t}) \end{array} \right\} \tag{E'3}$$
$$\mathcal{X}' = \{(\mathcal{M}, c, R) \mid \mathcal{M} \in \mathcal{B}\} \tag{X'1}$$
$$\cup \{(\mathcal{M}, c, R) \mid \boldsymbol{t}, \boldsymbol{t'} \in c; \boldsymbol{t} \neq \boldsymbol{t'}; \text{both enabled in } \mathcal{M}\} \tag{X'2a}$$
$$\cup \{(\mathcal{M}, c, R) \mid \boldsymbol{t} \in c; \text{enabled in } \mathcal{M}; 0 < \mathit{pre}(\boldsymbol{t})(\boldsymbol{p}) < \mathcal{M}(\boldsymbol{p}) \text{ for some } \boldsymbol{p} \in \mathcal{P}\} \tag{X'2b}$$
$$\cup \{(\mathcal{M}, c, R) \mid \text{Some } \boldsymbol{t} \in \mathcal{T} \text{ enabled; all such } \boldsymbol{t} \text{ involve the system and } \boldsymbol{t} \notin c\} \tag{X'3}$$

Figure 2 Description of the two graph games constructed from $\mathcal{G}$. For the components of $\mathit{Graph}(\mathcal{G})$, ignore all colored parts. Including them, we get the components of $\mathit{Graph}'(\mathcal{G})$.

4 Reduction to games over finite graphs

We wish to decide whether a k-bounded Petri game with one system player admits a winning, deadlock-avoiding strategy. In case of a positive answer, we also want to obtain a description of such a strategy. Note that the system player's decisions can be based on an unboundedly growing amount of information. Because of this, it is not at all obvious that the existence of a strategy is decidable and that strategies can be represented in finite space.

In this section and the next, we show that the decision problem is EXPTIME-complete in the size of the net. We establish the upper bound through a many-one reduction to a complete-observation game over a finite graph. We consider Petri games with a single system player, i.e., all reachable markings $\mathcal{M}$ contain exactly one system place, which we denote by $\boldsymbol{s_{\mathcal{M}}}$. In the cuts $\mathcal{C}$ of a strategy, we denote the unique system place by $s_{\mathcal{C}}$.

For a given Petri net $\mathcal{G}$ with underlying net $\mathcal{N}$, Fig. 2 defines the components of the translated graph game $\mathit{Graph}(\mathcal{G}) = (\mathcal{V}_0, \mathcal{V}_1, \mathcal{I}, \mathcal{E}, \mathcal{X})$ if we ignore all colored parts. The set of *vertices* $\mathcal{V}$ consists of two disjoint subsets $\mathcal{V}_0$ and $\mathcal{V}_1$, which describe the vertices belonging to players 0 and 1, respectively. The game begins in the initial vertex $\mathcal{I}$. From a vertex $v \in \mathcal{V}$, the current player chooses an outgoing *edge* in $\mathcal{E}$. A *play*, i.e., a maximal sequence $\mathcal{I} = v_0\, v_1\, \ldots$ of vertices with $(v_i, v_{i+1}) \in \mathcal{E}$ for all i, is winning for Player 0 if no vertex is an element of the bad vertices $\mathcal{X}$. A *strategy* T_σ (for Player 0) is a $\mathcal{V}$-labeled tree whose root is labeled with $\mathcal{I}$. If a node is labeled with a vertex in $\mathcal{V}_1$, its children are labeled with all successor vertices. Otherwise, it has a single child labeled with one particular successor. The strategy is *winning* if all maximal paths through it are labeled with winning plays. All such games are *memoryless determined*: If there is any winning strategy, there exists a winning strategy that selects, from any two nodes with the same label, the same successor vertex.

The vertices of the game essentially represent the reachable markings of the Petri game and Player 1 moves between markings by firing enabled transitions. This means that Player 1 plays the role of both the environment and the nondeterminism stemming from different schedulings. Player 0, who represents the system, can only act by refusing to allow some

transitions in the postcondition of the single system place in the marking. Since these decisions should not depend on scheduled, purely environmental transitions that the system would not yet know in the Petri game, Player 0 is forced to choose directly after the system player has taken a transition. Similarly to [12], we therefore add a *commitment*, i.e., a set $c \subseteq post(\boldsymbol{s_{\mathcal{M}}})$, to each vertex of the graph game. The commitment keeps track of the set of outgoing transitions of the current system place that the system player allows. Player 0's vertices are marked with $\top$ instead of a commitment to denote that she needs to decide on a commitment in the next step (E1). Player 1's choices are then restricted such that she can fire all purely environmental transitions (E2) but can only fire system transitions that appear in the commitment (E3). The bad vertices correspond to bad markings (X1), nondeterminism (X2a, X2b) and deadlock (X3).

To prove the reduction correct, we need to show that $\mathcal{G}$ has a winning, deadlock-avoiding strategy iff Player 0 has a winning strategy in $Graph(\mathcal{G})$. For this, we give translations between these types of strategies.

4.1 From Petri game strategies to graph game strategies

Assume that we are given a winning, deadlock-avoiding strategy $\beta_\sigma = (\mathcal{N}^\sigma, \lambda)$ for $\mathcal{G}$. We inductively build a strategy T_σ for $Graph(\mathcal{G})$. Whenever we encounter a node labeled with a vertex belonging to Player 0, we choose an outgoing edge, i.e., a suitable commitment.

For any such node, we look at the sequence of labels on the path that leads to it from the root. This sequence is a prefix of a play, which we denote by $v_0\, v_1\, \ldots\, v_r = (\mathcal{M}, \top)$. Edges of type (E1) in this prefix do not change the marking. All other edges are associated with firing a transition. Starting from the initial cut, we fire λ-preimages of these transitions one after another. If multiple transitions could be responsible for the edge or if multiple preimages are enabled, choose one canonically. For edges of type (E2), such preimages always exist because justified refusal does not allow β_σ to restrict purely environmental transitions. In the case of edges of type (E3), we make sure to only include transitions in the commitment if the existence of such preimages is ensured. By consecutively firing such a sequence of transitions, we reach a cut $\mathcal{C}$ such that $\lambda[\mathcal{C}] = \mathcal{M}$. Set $c \coloneqq \{\lambda(t) \mid t \in post(s_{\mathcal{C}})\}$ and choose the outgoing edge leading to $(\mathcal{M}, c)$ to construct the strategy.

For well-definedness, it remains to show that, when Player 1 schedules a system transition $\boldsymbol{t} \in c$ the next time, a preimage of this transition will be enabled in the cut $\mathcal{C}'$ that corresponds to the node in the strategy. Since, in between, only purely environmental transitions will be fired, $s_{\mathcal{C}}$ will still be part of $\mathcal{C}'$. The system place has a preimage of $\boldsymbol{t}$ in its postcondition by the definition of c. Therefore, a preimage enabled in $\mathcal{C}'$ exists by justified refusal.

▶ **Theorem 1.** *T_σ is a winning strategy for Player 0.*

Proof sketch (detailed in full version [11, B.2]). Consider a node n in T_σ with the label $(\mathcal{M}, c)$. As in the construction of the graph game strategy, we canonically fire transitions corresponding to the prefix until we reach a cut $\mathcal{C}$ such that $\lambda[\mathcal{C}] = \mathcal{M}$. Now assume that n is a bad vertex. Each kind of bad vertices (X1), (X2a), (X2b) or (X3) translates to a violation of the properties of a winning, deadlock-avoiding strategy in $\mathcal{C}$, contradiction. Thus, no node is labeled with a bad vertex and the strategy is winning. ◀

4.2 From graph game strategies to Petri game strategies

The converse direction is harder to prove. So far, we have shown that, if the system can win a Petri game with incomplete information, Player 0 can also win a game with full information on the marking graph. This is not surprising. In this step however, we must show that this

additional information does not give an advantage to Player 0 that the system does not have. In the construction of $\mathit{Graph}(\mathcal{G})$, we have already introduced commitments, which prevent Player 0 from using information about the scheduling of purely environmental transitions for her subsequent move. However, Player 0 might still use this information to make her move after the next. If the system player does not learn about the environment transition in her next step, this is an illegal flow of information.

The main idea now is that, while some parts of the graph game strategy do not correspond to a valid information flow in the Petri game, others do. In these latter parts, the strategy contains all necessary decisions to win the Petri game. Conceptually, we need to cut away unreasonable plays from the strategy. Alternatively, we might say that a forbidden information flow only happens if Player 1 does not play in an intelligent way. From Player 1's point of view, it is dangerous and unnecessary to schedule a purely environmental transition and then schedule a system transition unless the former is needed to enable the latter. If she does so, Player 0 gains potentially useful information, which Player 1 could easily prevent by scheduling the purely environmental transition at a later point, i.e., when it is necessary to enable the next system transition or when a winning situation for Player 1 (bad marking, nondeterminism or deadlock) can be reached without any more moves by Player 0. To make this idea formal, we construct another graph game $\mathit{Graph}'(\mathcal{G})$, which restricts Player 1's moves to enforce the behavior described above. Then, we can easily show that any winning strategy for $\mathit{Graph}(\mathcal{G})$ translates to a winning strategy for $\mathit{Graph}'(\mathcal{G})$, where Player 1 has fewer options. In a second step, we will translate the strategy from $\mathit{Graph}'(\mathcal{G})$ back to a strategy for the Petri game, which will prove the desired equivalence.

The new graph game $\mathit{Graph}'(\mathcal{G}) = (\mathcal{V}_0', \mathcal{V}_1', \mathcal{I}', \mathcal{E}', \mathcal{X}')$ is defined in Fig. 2 by taking into account the colored parts. The vertices of $\mathit{Graph}(\mathcal{G})$ are extended by a third component, a *responsibility multiset* R over $\mathcal{P}$. This multiset $R \subseteq \mathcal{M}$ tracks the information generated by firing transitions. At any point in the Petri game, a subset S of the cut such that $\lambda[S] = R$ together carries the information about all fired transitions. This notion is made precise in [11, Lemma 18]. After a transition has been fired, every token in its postcondition carries the information about the causal pasts of all participating tokens and about the fired transition itself. For this reason, when an edge of type (E'2) fires a purely environmental transition $\boldsymbol{t}$, the tokens in $\mathit{pre}(\boldsymbol{t})$ are subtracted from R, and Player 1 chooses an arbitrary token $\boldsymbol{o} \in \mathit{post}(\boldsymbol{t})$, which will carry the information to the system player. Edges of type (E'3) deal with R similarly in that they also subtract the precondition from R and instead add one element of the postcondition, namely the system place. In contrast to $\mathit{Graph}(\mathcal{G})$, these edges only allow system transitions if the responsibility multiset is included in the precondition, i.e., if the system player would directly learn about all previously scheduled transitions by taking this system transition.

▶ **Theorem 2.** *If there is a winning strategy for $\mathit{Graph}(\mathcal{G})$, there exists a winning strategy for $\mathit{Graph}'(\mathcal{G})$.*

Proof sketch (detailed in full version [11, B.3]). $\mathit{Graph}'(\mathcal{G})$ only reduces Player 1's options. ◀

We now translate a winning strategy T_σ for $\mathit{Graph}'(\mathcal{G})$ back into a winning, deadlock-avoiding strategy for the Petri game. Without loss of generality, we assume T_σ to be memoryless. We traverse the strategy tree in breadth-first order and inductively build the Petri game strategy $\beta_\sigma = (\mathcal{N}^\sigma, \lambda)$. Simultaneously, we map each node of the tree to a nonempty set of cuts. We call these cuts the *associated cuts* of the node. These cuts can be reached from In^σ by firing λ-preimages of transitions corresponding to the edges of types

(E'2) and (E'3) on the path from the root to this node. In particular, every such cut $\mathcal{C}$ will satisfy $\lambda[\mathcal{C}] = \mathcal{M}$, where $\mathcal{M}$ is the marking found in the label of the node.

We begin by mapping the root of the tree to a single cut In^σ, i.e., a fresh set of places such that $\lambda[In^\sigma] = In$. Then, we traverse T_σ and distinguish between the different kinds of edges in the graph game by which the vertex of the currently visited node has been reached from its predecessor.

- (E'1): Do not modify β_σ and map the new node to the same cuts as its parent.
- (E'2) or (E'3): Let $\mathcal{C}$ be one of the cuts associated with the parent node. Let $\boldsymbol{t}$ be a transition that could have been used in the definition of (E'2) or (E'3) to justify the existence of the edge. Finally, let B be any subset of $\mathcal{C}$ with $\lambda[B] = pre(\boldsymbol{t})$. Such a subset always exists because $\boldsymbol{t}$ is enabled in $\lambda[\mathcal{C}]$. If it already exists, let $t \in \mathcal{T}^\sigma$ be a transition with $pre(t) = B$ and $\lambda(t) = \boldsymbol{t}$. Else, create a new such transition and a fresh set of places as its postcondition such that $\lambda[post(t)] = post(\boldsymbol{t})$. Choose $\mathcal{C}'$ such that $\mathcal{C}\ |t\rangle\ \mathcal{C}'$. We map the new node to all cuts $\mathcal{C}'$ that can be constructed from suitable $\mathcal{C}$, $\boldsymbol{t}$ and B in this way.

We need to show that β_σ is a strategy. First, we can easily see that the construction ensures all requirements of an occurrence net. Furthermore, β_σ is an initial branching process because λ is an initial homomorphism and because we only add a new transition if no other transition with the same label and precondition exists.

Before we can prove that β_σ satisfies the four axioms of a winning, deadlock-avoiding strategy, we need to show that the responsibility multiset construction works as intended. First, we show that the construction prevents illegal information flows. Whenever the system player moves in the graph game, she directly learns about all previously scheduled transitions. Formally, nodes labeled with player-0 vertices are only mapped to cuts $\mathcal{C}$ that are the *last known cuts* of their respective system place $s_\mathcal{C}$. The last known cut of a place $x \in \mathcal{P}^\sigma$ is defined as $LKC(x) := \{p \in \mathcal{P}^\sigma \mid p \not< x \wedge \forall t \in pre(p).\ t < x\}$. In the terminology of [6], this cut is the *mapping cut* of $past(x) \cap \mathcal{T}$, i.e., the cut reached by firing all transitions in the past of x. The last known cut of x has the special property that, for every cut $\mathcal{C}$ with $x \in \mathcal{C}$, the last known cut of x lies in $past(\mathcal{C})$ [11, Lemma 20].

▶ **Lemma 3.** *Let a node in T_σ be labeled with a vertex belonging to Player 0 and let $\mathcal{C}$ be one of its associated cuts. Then, $\mathcal{C} = LKC(s_\mathcal{C})$.*

Proof in full version [11, B.4]. ◀

Second, we need to show that the responsibility multiset construction does not overly restrict the scheduling. For certain schedulings of purely environmental transitions, the responsibility multiset prevents a system transition from being fired even though it is enabled and in the commitment. If, since the Player 0's last move, Player 1 had skipped firing all transitions that do not help to enable this system transition, the transition could be fired. Therefore, the Petri game strategy contains all system transitions wherever they are not refused. This is formally stated and proved in [11, Lemma 21].

▶ **Lemma 4** (safety). *Let $\mathcal{C}$ be a cut in $\mathcal{N}^\sigma$. Then, $\lambda[\mathcal{C}] \notin \mathcal{B}$.*

Proof. Consider the node n for which $s_\mathcal{C}$ was inserted into the strategy. This node must be labeled with a $\mathcal{V}_0$ vertex and must have $LKC(s_\mathcal{C})$ as one of its associated cuts by Lemma 3. Since $LKC(s_\mathcal{C}) \subseteq past(\mathcal{C})$, there is a sequence of purely environmental transitions leading from $LKC(s_\mathcal{C})$ to $\mathcal{C}$ [11, Lemma 16]. Thus, from n's unique successor, we can follow a corresponding sequence of type-(E'2) edges to a node n' with $\mathcal{C}$ as one of its associated cuts. If $\lambda[\mathcal{C}]$ were a bad marking, n' would be labeled with a bad vertex of type (X'1). Since T_σ is a winning strategy, this is not the case. ◀

For the proofs of justified refusal (Lemma 24), determinism (Lemma 25) and deadlock avoidance (Lemma 26), we refer the reader to the full version [11]. As an immediate consequence, β_σ is a winning, deadlock-avoiding strategy, which concludes the claimed equivalence:

▶ **Theorem 5.** *If Graph′(𝒢) has a winning strategy for Player 0, there exists a winning, deadlock-avoiding strategy for 𝒢.*

5 Synthesis in distributed environments is EXPTIME-complete

▶ **Theorem 6.** *For fixed $k \geq 1$, k-bounded Petri games with one system player and an arbitrary number of environment players can be decided in exponential time.*

Proof. Our reduction allows to decide such Petri games $\mathcal{G}$ in exponential time: The number of vertices in $Graph(\mathcal{G})$ is bounded by $k^{|\mathcal{P}|} \cdot (2^{|\mathcal{T}|} + 1)$ and its local structure can be computed efficiently. Since graph games with such safety winning conditions can be solved in linear time in the size of the game [1, pp. 78–79], this requires exponential time in the size of the Petri game.

In [11, App. C.1], we describe an algorithm that evaluates the commitments symbolically and uses a SAT solver to speed up solving the game in practice. If we solve the SAT instances through a naïve enumeration, we have an explicit EXPTIME algorithm, whose complexity is analyzed in [11, App. C.2]. ◀

▶ **Theorem 7.** *Deciding k-bounded Petri games with one system player and an arbitrary number of environment players is EXPTIME-hard for any $k \geq 1$.*

Proof sketch (detailed in full version [11, B.7]). We show hardness through a reduction from the EXPTIME-complete combinatorial game G_5 from [20]. This reduction is similar to the one given in [12] for the fragment with one environment player. In G_5, two players, P_S and P_E, take turns in switching the truth values of a finite set of Boolean variables, one at a time. Alternatively, they are allowed to pass. The players operate on disjoint subsets of the variables. Initially, the variables have predefined values. If, at a certain point, a formula ϕ over the variables becomes satisfied, P_E wins; else, P_S wins.

For an instance of this game, we build a Petri game such that there is a winning, deadlock-avoiding strategy iff P_S has a winning strategy in the original game. Without loss of generality, let ϕ be given in negation normal form. An example for the reduction is illustrated in [11, Fig. 4]. Each variable is represented by an environment token moving between two places, indicating the variable's truth value. An additional environment token keeps track of the current turn. If it is P_E's turn, this token synchronizes with one of the environment variables and switches its position. If it is P_S's turn, the token first informs the single system token of the previous moves and then enables the transitions for switching a system variable, from which the system token chooses one.

Instead of letting a player move, the turn token can permanently freeze the variables and prove that ϕ is satisfied. For this, we have an additional environment token for every subformula, each with two places. The turn token can move these tokens to their second place to prove that the subformula is satisfied. For literals, the turn token needs to synchronize with the respective variable in the correct place. For disjunctions, it must synchronize with the token of one of the subformulas, which must have been proved before. For conjunctions, synchronization with both subformula tokens is required. The bad markings are exactly those in which the entire formula ϕ is proved. This game is 1-bounded, thus k-bounded. ◀

6 Sparse Petri games

The nets produced by our EXPTIME-hardness reduction contain a high number of environment tokens. Because of this, the number of reachable markings grows exponentially and computational cost with it. To study other sources of algorithmic hardness, we analyze the complexity of the problem for a fixed maximum number p of environment players. Then, we can bound the number of reachable markings by the polynomial $(|\mathcal{P}|+1)^{(p+1)}$ instead of by $(k+1)^{|\mathcal{P}|}$. For a fixed p, the problem is in NP: We nondeterministically guess a commitment for every $\mathcal{V}_0$ vertex and verify in polynomial time that no bad vertices are reachable.

▶ **Theorem 8.** *For a fixed $p \geq 3$, deciding Petri games with one system player and p environment players is NP-complete.*

Proof sketch (detailed in full version [11, B.8]). The upper bound has already been established. Show the lower bound by a reduction from the Boolean satisfiability problem with 3-clauses (3SAT). For a given instance, construct a Petri game with three environment players and a single system player. For every clause, the single system player must allow at least one transition corresponding to a satisfied literal in the clause. Deadlock avoidance forces the system player to allow at least one such transition per clause. Nondeterminism prevents the system player from allowing two transitions corresponding to complementary literals. ◀

▶ **Theorem 9.** *Petri games with one system player and at most two environment players can be decided in polynomial time.*

Proof sketch (detailed in full version [11, B.9]). We adapt the algorithm in [11, App. C.1], which evaluates commitments symbolically with a SAT solver. Due to the special structure of the SAT instances generated, we can add pre- and postprocessing steps such that the SAT queries only contain 2-clauses. Since 2SAT can be solved in polynomial time [2], this yields a polynomial-time decision procedure. ◀

7 Conclusions

In this paper, we have developed algorithms for the synthesis of reactive systems in distributed environments. We have studied the problem in the setting of Petri games. Previously, the decidability of Petri games was only known for non-distributed environments, i.e., for games with a single environment token [12]. Our algorithms solve Petri games with one system token and an arbitrary number of environment tokens. We have shown that the synthesis problem can be solved in polynomial time for nets with up to two environment tokens. For an arbitrary but fixed number of three or more environment tokens, the problem is NP-complete. If the number of environment tokens grows with the size of the net, the problem is EXPTIME-complete.

An intriguing question for future work is whether our results, which scale to an arbitrary number of environment tokens, can be combined with the results of [12], which scale to an arbitrary number of system tokens. This would allow us to synthesize "distributed systems in distributed environments." With the algorithm presented in this paper, we can already synthesize individual components in such distributed systems, by treating the other components as adversarial (cf. [13]). The approach of [12] would additionally allow us to analyze the cooperation between the system components.

References

1 Krzysztof R. Apt and Erich Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

2 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

3 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 652–657. Springer, 2012. doi:10.1007/978-3-642-31424-7_45.

4 J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. doi:10.2307/1994916.

5 Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6605 of *Lecture Notes in Computer Science*, pages 272–275. Springer, 2011. doi:10.1007/978-3-642-19835-9_25.

6 Javier Esparza. Model checking using net unfoldings. *Sci. Comput. Program.*, 23(2-3):151–195, 1994. doi:10.1016/0167-6423(94)00019-0.

7 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017. doi:10.1007/978-3-662-54577-5_20.

8 Bernd Finkbeiner. Bounded synthesis for petri games. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2015. doi:10.1007/978-3-319-23506-6_15.

9 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Symbolic vs. bounded synthesis for Petri games. In Dana Fisman and Swen Jacobs, editors, *6th Workshop on Synthesis (SYNT 2017)*. EPTCS, 2017. URL: https://www.react.uni-saarland.de/publications/FGHO17.pdf.

10 Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-based synthesis of distributed systems. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 433–439. Springer, 2015. doi:10.1007/978-3-319-21690-4_25.

11 Bernd Finkbeiner and Paul Gölz. Synthesis in distributed environments. *CoRR*, abs/1710.05368, 2017. arXiv:1710.05368.

12 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Inf. Comput.*, 253:181–203, 2017. doi:10.1016/j.ic.2016.07.006.

13 Bernd Finkbeiner and Sven Schewe. Semi-automatic distributed synthesis. In Doron A. Peled and Yih-Kuen Tsay, editors, *Automated Technology for Verification and Analysis,*

Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7, 2005, Proceedings, volume 3707 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2005. doi:10.1007/11562948_21.

14 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

15 Swen Jacobs, Roderick Bloem, Romain Brenguier, Ayrat Khalimov, Felix Klein, Robert Könighofer, Jens Kreber, Alexander Legg, Nina Narodytska, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The 3rd reactive synthesis competition (SYNTCOMP 2016): Benchmarks, participants & results. In Ruzica Piskac and Rayna Dimitrova, editors, *Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016.*, volume 229 of *EPTCS*, pages 149–177, 2016. doi:10.4204/EPTCS.229.12.

16 Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 258–262. Springer, 2007. doi:10.1007/978-3-540-73368-3_29.

17 Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 639–651. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

18 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

19 Stuart Russell and Peter Norvig. *Artificial Intelligence – A Modern Approach*. Pearson, 3rd edition, 2009.

20 Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM J. Comput.*, 8(2):151–174, 1979. doi:10.1137/0208013.

21 Wieslaw Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.

Train Scheduling on a Unidirectional Path

Apoorv Garg[1] and Abhiram G. Ranade[2]

1 **Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India**
apoorv.garg@gmail.com

2 **Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India**
ranade@cse.iitb.ac.in

Abstract

We formulate what might be the simplest train scheduling problem considered in the literature and show it to be NP-hard. We also give a log-factor randomised algorithm for it. In our problem we have a unidirectional train track with equidistant stations, each station initially having at most one train. In addition, there can be at most one train poised to enter each station. The trains must move to their destinations subject to the constraint that at every time instant there can be at most one train at each station and on the track between stations. The goal is to minimise the maximum delay of any train. Our problem can also be interpreted as a packet routing problem, and our work strengthens the hardness results from that literature.

1998 ACM Subject Classification F.2.2 Sequencing and scheduling, G.2.1 Combinatorial algorithms, G.2.2 Network problems

Keywords and phrases Combinatorial optimisation, Train scheduling, Max-delay minimisation, Complexity analysis, Approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.29

1 Introduction

In this paper, we formulate a very simple train scheduling problem, show its NP-hardness, and give a randomised log-factor approximation algorithm.

Train scheduling is an extensively researched area (see, for example, the recent surveys of [1, 7, 19, 21]). The major model used is as follows. We are given a graph in which vertices represent stations and edges represent tracks. Initially, each station may hold one or more trains, which are to be moved to specified stations using specified paths. Under the standard signalling regime, on each edge there can be at most one train, and it takes some specified finite time for a train to cross an edge. Additionally, there exists a buffering limitation – each station is capable of holding no more than a specified number of trains. The goal is to move the trains such as to minimise the *makespan* (maximum completion time), *flow-time* (total completion time), or *max-delay* (maximum delay suffered by any train). This abstract problem is also studied in the packet routing literature with trains, stations, and tracks replaced by packets, network nodes, and communication links.

The problem as defined above is known to be NP-complete in various versions. See [20] for minimising the makespan, assuming unbounded buffering at each node, even when the graph is a tree. See [5] for minimising the makespan or max-delay even to a constant factor, assuming bounded buffering at nodes, for levelled directed networks in which packets move from the lowest numbered level to the largest numbered level.

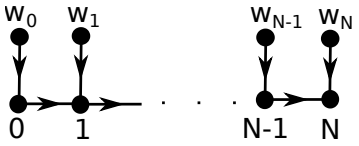


Figure 1 *The model network*

For general networks, constant factor approximation algorithms are known for minimising the makespan even with constant number of buffers at each node [14, 15, 24, 22]. For directed as well as undirected trees a 2-approximation of the optimal makespan can be obtained if we have an unbounded number of buffers in each node [20]. For unidirectional rings, in-trees, and out-trees, the optimal makespan can be obtained with unbounded buffers [16]. One recent study analyses the computational complexity of a specific train scheduling problem [10]. It models the problem of scheduling, with minimum makespan, several trains from opposite sides along a single bi-directional track with unbounded capacity at intermediate stations where trains can pass and cross each other. The problem is shown to be pseudo-polynomially solvable with equal train-speeds; however, with different speeds, it can be translated to a job-shop scheduling problem that is already strongly NP-hard.

Our concern in this paper is scheduling trains on a directed path, with trains allowed to enter or exit the path at any station. We consider minimising the max-delay, which we believe is more appropriate for train scheduling. The motivation for our study is twofold. First, large train networks are often broken into smaller networks for the purpose of managing them. These smaller networks often consist of a major trunk route with trains entering and exiting the route from and to branch lines. Each direction of the route is like the path network we consider. In addition, we are interested in a path network also because it is presumably the simplest network possible. Indeed, we further simplify the network – we assume that the inter-station distances, as well as the train speeds, are identical. We feel that we should figure out good theory for this elementary model before moving on to more complex ones. Finally, we note that there is also a large amount of experimental work on train scheduling using simulation, heuristics, integer linear programming, game theory, etc. [2, 3, 4, 6, 8, 11, 12, 17, 18, 23]. Our interest, as explained earlier, is different.

Outline of the paper is as follows. In Section 2, we formally define our problem. We also define in Section 3 a *chain-hole view* of the movement of trains, which is useful in the exposition of our lower and upper bounds. In Section 4, we show that finding a schedule with minimum max-delay is NP-hard. In Section 5, we present a randomised algorithm that schedules trains such that the max-delay is within a log-factor of the optimal. Section 6 concludes with directions for future work.

2 The Train Scheduling problem

We consider the network shown in Figure 1. It consists of :

1. The *line* – a sequence of $N+1$ stations labelled $0, 1, \ldots, N$, and unit-length directed links $(s, s+1)$ connecting every pair of consecutive stations s and $s+1$.
2. The *branches* – an *outer* (where a train waits before entering the line) w_s corresponding to each station s, and a unit-length link (w_s, s) connecting w_s to s.

Every station and outer has a capacity to hold at most one train at a time. A train takes unit time to move from one station to the next, or from an outer to its corresponding station. When there is no train at a station s, we say that there is a *hole* at s. When a train reaches

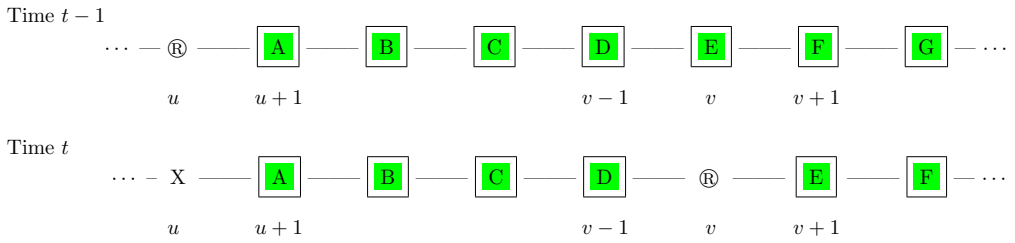


Figure 2 *Example of a hole-jump*
Boxes represent trains, circles represent holes. At time $t-1$, there is a hole R at station u. During step t, train E moves from station v to $v+1$, while the trains A, B, C, and D wait at their respective stations $u+1$ to $v-1$. The hole thus appearing at station v at time t is considered to be the same as was present at station u at time $t-1$. We say that the hole jumped from u to v in step t.

its destination, it immediately *vanishes* (exits), leaving a hole at that station. Every train that is initially at a station is called an *internal train*, while every train that is initially at an outer is called an *external train*.

Path of a train consists of all nodes and links it visits during its journey, including the origin and the destination. Note that an external train has to *enter* the line (i.e., move from the outer to its corresponding station) before it can move on the line towards its destination. Therefore, while path-length of an internal train is just the distance from its origin to its destination, that of an external train is one unit more than the distance from its entry station to its destination. The event of an external train entering the line will be referred to as an *entry*, to distinguish it from a *movement* which will refer to the event of a train moving on the line from one station to the next.

An instance is defined by specifying (i) the number N, and (ii) destination of the train, if any, placed initially at each station and outer; the destination must of course be downstream of the initial position. In any schedule for movement of the trains to their destinations, the amount of time a train remains stationary is said to be its *delay*. The required output is a schedule such that *max-delay* – the maximum among the delays – is minimised.

The t^{th} *step* of the schedule denotes the unit time duration $(t-1, t]$. The *entry time* of an external train is the time when it entered the line. *Last-entry-time* of the schedule is the last time instant when some external train entered the line, i.e., the maximum among the entry times of external trains. Without loss of generality, we assume that after the last-entry-time, all the trains proceed to their destinations without further delays.

▶ **Theorem 1.** *In any schedule,* $(\textit{last-entry-time} - 1) \leq \textit{max-delay} \leq \textit{last-entry-time}$

Proof. No train waits after the last-entry-time, say T. Hence, the max-delay can be at most T. If an external train has suffered the maximum delay, then the max-delay is $T-1$. If every train that suffered the maximum delay is an internal train, then the max-delay is T. ◀

Hence, from now on, we will worry about minimising the last-entry-time.

3 Chain-Hole view of schedules

An external train can only enter a hole. Thus, it is useful to understand holes :

Pre-existing holes. Initially, as part of the input, we are given some *pre-existing* holes on the line. In addition, it will be convenient to assume that we also have stations $-1, -2, \ldots$ upstream of station 0, each having a pre-existing hole; we will call them as *external holes*. Clearly, these *imaginary* stations and holes cannot affect the movement of trains.

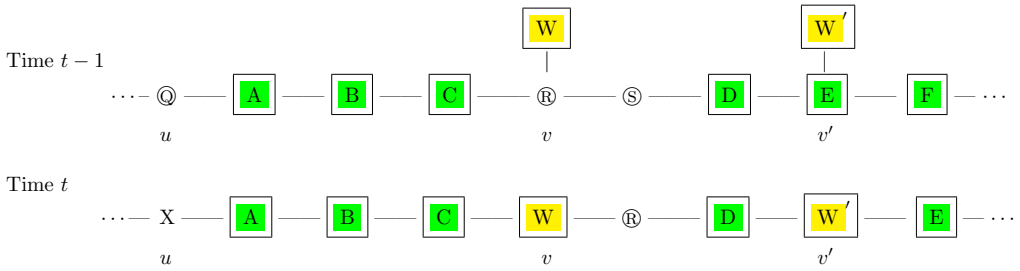


Figure 3 *External trains enter by filling holes*
In step t, hole Q jumps across four links, from station u to station $v = u + 4$, and is immediately filled by an external train W that enters the line at station v simultaneously with the jump of Q. In the same step, another external train W′ enters the line at a station v', filling a hole S which has simultaneously jumped to v' from station $v' - 2 = v + 1$. Hence, the trains A, B, C, and D have to wait during this step, while the trains E and F as well as the hole R must move.

Hole creation. A hole is created when a train moves into its destination station and vanishes.
Hole destruction. A hole is destroyed when it is *filled* by an external train.
Hole movement (jump). Suppose at time $t-1$ a hole is present in station u. Suppose that the stations $u+1, \ldots, v-1$ have trains which do not move in step t, while the train at station v moves. Then we will say that the hole at u at time $t-1$ has jumped to v in step t. Similarly, if the trains at stations $0, 1, \ldots, s-1$ wait and the train at s moves, we consider that to be a jump of an external hole from station -1 to station s.

Note that in a single step, a hole can jump across any number of links, while a train can only move across one link (to an adjacent station). However, hole-jumps and train-movements cannot overlap, as delineated in Lemma 2, which follows from the above definitions.

▶ **Lemma 2.** *If two holes jump in the same step, then the paths of those jumps must be link-disjoint, i.e., they do not share any link. If a train-movement occurs in the same step as a hole-jump, then their paths too must be link-disjoint.*

3.1 Chains induced by a schedule

In any given schedule, consider an external train p that enters the line by filling a hole $\overline{h}$. Define $\overline{h}$ as the predecessor of p. If $\overline{h}$ is created by the exit of another train $\overline{p}$, then define $\overline{p}$ as the predecessor of $\overline{h}$. This procedure will link every external train into an alternating sequence of trains and holes. Each such maximal sequence is called a *chain*.

The first element of a chain can either be an internal train or a pre-existing hole, both of which do not have predecessors. That train or hole is said to be the *beginning* of the chain. Let c be a chain and station s_0 be the initial position of its beginning. Let p_k be the last train of c, and s_k the entry station of p_k. Then the *span* of chain c is defined as the interval $[s_0, s_k]$. The last train p_k is said to be the *terminal train* of the chain, while all other trains of the chain are said to be its *non-terminal trains* (Figure 4). Every link within the span of a chain c is either run over by a non-terminal train of c, or jumped over by a hole of c. We say that the chain *crosses* all these links. Note that the links run over by the terminal train of a chain are, by definition, not crossed by the chain.

▶ **Definition 3.** *Congestion produced in a link*, say l, by a set $\mathcal{C}$ of chains is defined as the number of those chains in $\mathcal{C}$ which cross l. *Congestion of a set* of chains is the maximum congestion produced by the set in any link (including those upstream of station 0).

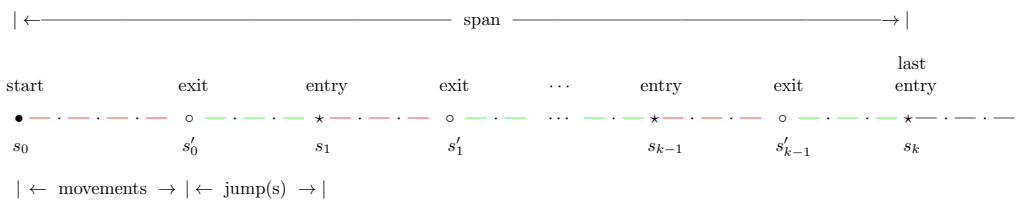


Figure 4 *Spatial view of a chain* $\langle p_0, h_0, p_1, h_1, \ldots, p_{k-1}, h_{k-1}, p_k \rangle$
Station s_0 is the origin (•) of the internal train p_0 with which the chain begins. Stations $s_1, \ldots, s_{k-1}, s_k$ are the entry points ($\star$) of external trains $p_1, \ldots, p_{k-1}, p_k$. Stations $s'_0, s'_1, \ldots, s'_{k-1}$ are the destinations (○) of non-terminal trains $p_0, p_1, \ldots, p_{k-1}$, where the holes $h_0, h_1, \ldots, h_{k-1}$ get created when those trains exit. Links (–) crossed by train-movements are shown in red, while the links (–) crossed by hole-jumps are in green. Trains have not been explicitly depicted.

▶ **Definition 4.** We define the *age* of a chain c as the minimum possible time by which its terminal train, say p, can enter the line, i.e., the entry time t of p if it were the case that in every step before t, either a train of c moves on the line or a hole of c jumps and gets filled in.

In other words, the age of c equals the number of movements on the line to be made by its non-terminal trains, plus the number of its holes, since every hole h of a chain makes at least one jump – the one coinciding with the entry of the train that succeeds h in the chain. Note that *the age of a chain is a unit more than the sum of path-lengths of its non-terminal trains.*

▶ **Theorem 5.** *Let $\mathcal{S}$ be a schedule, $\mathcal{C}$ the set of chains induced by $\mathcal{S}$, and T the last-entry-time in $\mathcal{S}$. Then $\mathcal{C}$ has a congestion at most T, and every chain in $\mathcal{C}$ has an age at most T.*

Proof. At most one train can cross a link in a single step. When a hole jumps in a step t from a station s to another station s', all trains and holes at the stations between s and s' halt during step t. Hence, for any link l, at most one train or hole can move or jump across l in each step, which implies that at most T chains can cross l by time T. Since the terminal trains of all chains have entered the line by time T, no chain crosses any link afterwards. Thus, congestion of any link can not exceed T, implying that $\mathcal{C}$ has a congestion at most T.

For any chain, movements of its trains along the line must happen at distinct times. Furthermore, the hole-jumps must also happen at distinct times and every hole must make at least one jump. Hence, the last hole-jump, which coincides with the entry of the last train of the chain, cannot be made at a time smaller than the age of the chain. ◀

4 NP-hardness

For the decision version of Train Scheduling, the input is as given in Section 2 together with an integer T. We are required to decide if there is a schedule $\mathcal{S}$ with max-delay less than T.

▶ **Theorem 6.** *Train Scheduling is NP-hard.*

Proof. The reduction is from the Bin Packing problem [9], for which the input is (i) a finite set $U = \{X_1, \ldots, X_n\}$ of positive integers, (ii) an integer bin capacity B such that $B \geq X_i \ \forall i$, and (iii) a positive integer M. The required decision is whether a partition of U into M disjoint subsets $U_1, \ldots, U_M$ exists such that the sum of integers in every subset is at most B.

Given a Bin Packing instance, our Train Scheduling instance is as follows. We have a line with stations $0, \ldots, N$ along with an integer T, where $N = \alpha(3+(n+1)B)$, $T = \alpha(B+1)$, and $\alpha = (n + M)$. Stations $T + 1, \ldots, T + M$ have holes $h_1, \ldots, h_M$; other stations have internal trains going to the last station N. For external trains, we first have T trains $p_1, \ldots, p_T$ at the outers of stations $0, \ldots, T - 1$, each going to station N. We then have n external trains

$q_1, \ldots, q_n$, each q_i going a distance αX_i. These wait, in any order, at the outers of stations downstream of station $T + M$ such that their paths are disjoint, i.e., they share neither a link nor a station. Let D be the most downstream station among the destinations of q_is. We also have M external trains $r_1, \ldots, r_M$ at the outers of stations $D + 1, \ldots, D + M$, each going to station N. It should be clear that for our chosen value of N, all these trains do fit within the network.[1] To complete the proof, we make the following three observations.

First, note that the time taken for the reduction is polynomial in n and B. Since Bin Packing is strongly NP-hard, we may assume its input to be in unary. Hence, the reduction runs in time polynomial in the size of the Bin Packing instance.

Second, suppose the Bin Packing instance has a solution $\{U_1, \ldots, U_M\}$. Then, for the Train Scheduling instance, we can build a schedule with max-delay less than T as follows. We partition the external trains into $T + M$ chains. For every $j \in \{1, \ldots, T\}$, we construct a chain c_j consisting of (i) the external hole at station $-j$, and (ii) the train p_j. For every $k \in \{1, \ldots, M\}$, we construct a chain c'_k consisting of (i) the hole h_k, (ii) $|U_k|$ trains (and the holes created by their exits) corresponding to the integers in U_k, and (iii) the train r_k. Note that the age of every c'_k is $1 + \sum_{X \in U_k} \alpha X \leq 1 + \alpha B$, while the age of every c_j is 1.

In each step $j \in \{1, \ldots, T\}$ we schedule the train p_j to enter the line. Note that the other $(n + M)$ entries further downstream the line do not conflict with these T entries. Therefore, to prove that none of the external trains gets delayed by more than $T - 1$ steps, it suffices to show that the other entries can also be scheduled to take place by time T. In fact, we show that they can be made to take place by time $T - 1$ as follows. In each step $k \in \{1, \ldots, M\}$ we schedule the entry of the first train of chain c'_k. Subsequently, in every step we prioritise entries over movements on the line. There can be at most $n + M$ steps in which the trains q_i and r_k enter the line. In other steps, for each c'_k, one of its non-terminal trains moves on the line unless its last train r_k has already entered; there can be at most $\alpha B - 1$ such steps. Thus, the total number of steps by the time every r_k has entered must be at most $n + M + \alpha B - 1 = T - 1$. Hence, the maximum delay for the external trains is $T - 1$. This can be easily seen to hold also for the internal trains.[2]

Third, suppose the Train Scheduling instance has a schedule $\mathcal{S}$ with max-delay at most $T - 1$. Then we can build a solution for the Bin Packing instance as follows. From Theorem 1, it follows that the last-entry-time in schedule $\mathcal{S}$ can at most be T. Consider the set $\mathcal{C}$ of chains induced by $\mathcal{S}$. Suppose there are more than $T + M$ chains in $\mathcal{C}$. Since all internal trains go till N, every chain must begin with a hole. Since there are only M internal holes, more than T chains must begin with external holes, implying a congestion more than T in the link $(-1, 0)$, which is a contradiction to Theorem 5. Hence, $\mathcal{C}$ has at most $T + M$ chains. Each p_j goes till the end and therefore must be the terminal train of its chain. Moreover, it cannot have any other train in its chain since no other train ends upstream of its entry station. Therefore, just p_js take T chains. Hence, the other $n + M$ external trains must be packed in the remaining chains, which then must be M in number since each r_k has to be the terminal train of one, say c'_k. From Theorem 5, age of every c'_k is at most $T = \alpha(B + 1)$. That means the sum of path-lengths of all non-terminal trains of c'_k is less than $\alpha(B + 1)$, which implies that the sum of integers corresponding to the non-terminal trains in c'_k is strictly less than $B + 1$, i.e., at most B. Let U_k be the set of those integers. Thus, we get the required partition $\{U_1, \ldots, U_M\}$ of the input set U. ◀

---

[1] $D+M \leq \left(T + M + \sum_{i=1}^{n} \alpha X_i\right)+M \leq (\alpha(B+1) + M + n\alpha B)+M = \alpha+2M+\alpha(n+1)B < N$

[2] The internal trains initially at stations $0, \ldots, T$ always move ahead during the first T steps while p_js are entering the line. None of the internal trains initially at stations $(T + M + 1), \ldots, N$ halts after time $T - 1$ by when all q_is and r_ks have entered the line. No train, internal or external, halts after time T.

5 A log factor approximation

We present a randomised approximation algorithm which builds a schedule achieving a last-entry-time $O(T^* \log N)$ with high probability (w.h.p.), where T^* is the optimal last-entry-time. Theorem 1 implies that it is also a log-factor approximation for minimising max-delay. The algorithm consists of two procedures :

1. The *partitioning* procedure takes as argument a target T. If T is a feasible last-entry-time then it returns a set of chains having $O(T)$ age and $O(T)$ congestion, otherwise it correctly declares T to be infeasible.
2. The *scheduling* procedure uses the set of chains returned by the partitioning procedure to generate a schedule having a last-entry-time $O(T \log N)$ w.h.p.

The overall algorithm runs in two stages. In the first stage, by performing a binary search on T in the range 1 through N, it finds the smallest value $\tilde{T}$ for which the partitioning procedure returns a set of chains. In the second stage, it invokes the scheduling procedure with the set of chains obtained for $\tilde{T}$ to get a schedule $\tilde{S}$. Since we know that no schedule is possible with last-entry-time less than $\tilde{T}$, the schedule $\tilde{S}$ – guaranteed to have last-entry-time $O(\tilde{T} \log N)$ – is a $\log N$ approximation of the optimal. Sections 5.1 and 5.2 give the details.

5.1 Partitioning

The partitioning procedure is given in Algorithm 1. It is called with a target time T. We use the term *short* or *long* for a train to denote whether its path-length is less than or at least T.

5.1.1 Analysis

We will compare the chains in $\hat{\mathcal{C}}$, as they get constructed by the procedure, with the chains in the set $\mathcal{C}^*$ induced by an optimal schedule, i.e., a schedule having a last-entry-time T. The comparison will show that at every station the set $\hat{\mathcal{C}}$ has more active chains – to which the external train (if any) can be added – than $\mathcal{C}^*$. Additionally, it will show that the active chains of $\hat{\mathcal{C}}$ have more capacity to accommodate external trains than the active chains of $\mathcal{C}^*$. This, in turn, will imply that if T is a feasible last-entry-time then for every external train the procedure has an active chain to add the train to, and hence it does not abort; rather, it runs to completion and returns the set of chains. In the following, we call a chain a *short chain* (*long chain*) if it ends with a short (long) train.

▶ **Definition 7.** We define the *weight* of a chain as the sum of the path-lengths of all short trains (terminal train, if short, as well as the non-terminal trains) in the chain.

▶ **Corollary 8.** *Given a target T for last-entry-time, let $\mathcal{C}^*$ be the set of chains induced by an optimal schedule, i.e., a schedule having a last-entry-time T. Then every chain in $\mathcal{C}^*$ has a weight at most $2T - 2$, and every non-terminal train in $\mathcal{C}^*$ is a short train.*

Before embarking on the analysis, we make some technical modifications to $\mathcal{C}^*$ as follows :

1. If $\mathcal{C}^*$ has $x < T$ chains beginning with external holes, then we additionally include $T - x$ *degenerate*[3] chains, each containing an external hole not already there in another chain.

[3] A chain consisting of just a hole or a short internal train (no external trains) is called a *degenerate* chain.

Algorithm 1 The partitioning procedure

1: Initialise set $\hat{\mathcal{C}}$ with $9T$ chains beginning with external holes at stations $-1, \ldots, -9T$.
2: Designate every chain in $\hat{\mathcal{C}}$ as *active*.
3: **for** each station $s = 0, 1, \ldots, N$ **do**
4: **if** exists a short external train p at station s **then**
5: **if** exist active chains with weight $< 5T$ and last train ending upstream of s **then**
6: Add train p to any such chain c.
7: **else**
8: Declare T as infeasible and abort.
9: **end if**
10: **else if** exists a long external train p at station s **then**
11: **if** exist active chains with last train ending upstream of s **then**
12: Let c be any such chain with maximum weight.
13: Add train p to chain c.
14: *Terminate* chain c. {i.e., c is no longer active}
15: **else**
16: Declare T as infeasible and abort.
17: **end if**
18: **end if**
19: **if** exists a hole h or a short internal train p at station s **then**
20: **if** the number of active chains is $9T$ **then**
21: Let $\bar{c}$ be any active chain with maximum weight.
22: Terminate chain $\bar{c}$. {i.e., $\bar{c}$ is no longer active.}
23: **end if**
24: Add to $\hat{\mathcal{C}}$ a new chain c' beginning with h or p.
25: Designate c' as active.
26: **end if**
27: **end for**
28: Return $\hat{\mathcal{C}}$.

2. For every station $s \in \{0, \ldots, N\}$ having a hole h or a short internal train p, if $\mathcal{C}^*$ does not contain any chain beginning at s, then we add a degenerate chain containing h or p.
3. We extend the spans of short chains, without increasing the congestion of $\mathcal{C}^*$ beyond T, as follows. Let $[s, s']$ be the original span[4] of any chain $c \in \mathcal{C}^*$. Then the span of c is extended to $[s, s'']$, where s'' is as follows. If c is long, then $s'' = s'$; otherwise, s'' is the maximally downstream station from s' such that the congestion of $\mathcal{C}^*$ does not exceed T.

A chain $c \in \mathcal{C}^*$ that has an extended span $[s, s'']$ is said to *begin* at station s, be *active* at all stations and on all links in the open interval (s, s''), and be *terminated* at station s''. We say that $\mathcal{C}^*$ has been *maximally extended* by making these modifications. Note that this does not change the trains, holes, age, and weight of any chain already in $\mathcal{C}^*$.

▶ **Theorem 9.** *If the specified target time T is feasible, then the partitioning procedure completes successfully. Moreover, the set $\hat{\mathcal{C}}$ of chains it returns has a congestion at most $9T$, and each chain in the set has an age less than $6T$.*

Proof. First note that whenever the set $\hat{\mathcal{C}}$ has $9T$ active chains, the procedure terminates an active chain (Algorithm 1, line 20) before adding a new one (line 24). This implies that no

[4] The original span of a degenerate chain beginning at station s is taken to be $[s, s]$.

more than $9T$ chains are active on any link, i.e., the congestion of $\hat{\mathcal{C}}$ is at most $9T$. Moreover, once the weight of a chain exceeds $5T$, no more short trains are added to it. Hence, the weight of any chain can at most be $(5T-1)+(T-1) = 6T-2$, and its age at most $6T-1 < 6T$.

Now suppose that T is a feasible last-entry-time. Then there must exist an *optimal schedule* $\mathcal{S}^*$ achieving it. Let $\mathcal{C}^*$ be the maximally extended set of chains induced by $\mathcal{S}^*$. For any set $\mathcal{C}$ of chains – particularly for $\mathcal{C} \in \{\hat{\mathcal{C}}, \mathcal{C}^*\}$ – let $X(\mathcal{C}, s)$ denote the number of chains active on the link $(s-1, s)$, and let $W(\mathcal{C}, s)$ denote *the total weight of active chains before* s, i.e., the sum of path-lengths of all those short trains that originate upstream of s and belong to the chains active on $(s-1, s)$. We will prove that the following hold at each station $s \in \{0, 1, \ldots N\}$:

Invariant I. $\Delta X(s) := X(\hat{\mathcal{C}}, s) - X(\mathcal{C}^*, s) = 8T$

Invariant II. $\Delta W(s) := W(\hat{\mathcal{C}}, s) - W(\mathcal{C}^*, s) < 26\,T^2$

The two invariants will imply that $\hat{\mathcal{C}}$ has an active chain to which the external train (if any) at s can be added, and hence the procedure does not abort (line 8 or 16).

The proof is by induction on stations. At station 0, the invariants clearly hold. Suppose they hold at stations $0, \ldots, s$. Then, to prove them at station $s+1$, we consider all the cases :

1. *No external train, but a long internal train p at s*

   In $\hat{\mathcal{C}}$ as well as $\mathcal{C}^*$, neither any train is added nor any chain begins at s. In $\hat{\mathcal{C}}$, by construction, no chain is terminated at s, implying $X(\hat{\mathcal{C}}, s+1) = X(\hat{\mathcal{C}}, s)$ and $W(\hat{\mathcal{C}}, s+1) = W(\hat{\mathcal{C}}, s)$. In $\mathcal{C}^*$, since $X(\mathcal{C}^*, s) \le T$ and no chain begins at s, termination of a chain at s would mean that its extended span is not maximal – a contradiction to that $\mathcal{C}^*$ is maximally extended. Hence, in $\mathcal{C}^*$ as well no chain is terminated at s, that is, $X(\mathcal{C}^*, s+1) = X(\mathcal{C}^*, s)$ and $W(\mathcal{C}^*, s+1) = W(\mathcal{C}^*, s)$. Clearly, both the invariants hold at $s+1$.

2. *No external train, but a hole h or a short internal train p at s*

   In $\hat{\mathcal{C}}$ as well as $\mathcal{C}^*$, a chain begins at s. If $X(\hat{\mathcal{C}}, s) = 9T$, then from Invariant I, $X(\mathcal{C}^*, s) = T$, and hence a chain is terminated at s in each set (implied for $\hat{\mathcal{C}}$ by construction, and for $\mathcal{C}^*$ by congestion $\le T$). Otherwise, no chain is terminated in either (by construction of $\hat{\mathcal{C}}$, and by $\mathcal{C}^*$ being maximally extended). Both ways, Invariant I holds at $s+1$.

   For Invariant II, we first note that in each of the two sets, the chain beginning at s with h or p contributes the same additional weight – 0 (if it is h) or some $\ell < T$ (if it is p) – to the total weight of active chains before $s+1$. If no chain is terminated at s, then Invariant II clearly holds at $s+1$. Otherwise, we need to consider two subcases :

   a. Suppose in $\hat{\mathcal{C}}$, the chain terminated at s has weight $2T$ or more, then $\Delta W(s+1)$ can only be smaller than $\Delta W(s)$, since in $\mathcal{C}^*$ the weight of the chain terminated at s can at most be $2T-2$. Therefore, Invariant II holds at $s+1$.

   b. Suppose in $\hat{\mathcal{C}}$, the chain terminated at s has weight less than $2T$. Then, since it has maximum weight among all chains active on $(s-1, s)$, each of the other active chains must also have a weight less than $2T$. The same chains are also active on $(s, s+1)$ with same weights before $s+1$ as before s. The only additional chain active on $(s, s+1)$ is the one beginning at s and having weight 0 or $\ell < T$. Then $X(\hat{\mathcal{C}}, s+1) \le 9T$ implies $W(\hat{\mathcal{C}}, s+1) < 18\,T^2$. Since $W(\mathcal{C}^*, s+1) \ge 0$, Invariant II holds at $s+1$.

3. *A short external train p', and a long internal train p at s*

   Since in $\mathcal{C}^*$ the train p' must belong to some chain active on $(s-1, s)$, $X(\mathcal{C}^*, s) \ge 1$. Then Invariant I implies $X(\hat{\mathcal{C}}, s) \ge 8T+1$. Moreover, $X(\mathcal{C}^*, s) \le T$ since the congestion of $\mathcal{C}^*$ is at most T, and from Corollary 8 the weight of any chain in $\mathcal{C}^*$ is at most $2T-2$. Hence, $W(\mathcal{C}^*, s) < 2T^2$, and by Invariant II, $W(\hat{\mathcal{C}}, s) < 26\,T^2 + 2T^2 = 28\,T^2$. In $\hat{\mathcal{C}}$, out of

all chains active on $(s-1, s)$, only less than $2T$ may end at or downstream of s.[5] Thus, more than $6T$ active chains must end upstream of s. Then for the average weight, say $\overline{W}$, of upstream ending active chains, we have $\overline{W} < \frac{28\,T}{6} < 5T$. Since the average weight before s of the upstream ending active chains is less than $5T$, at least one of these chains must have weight less than $5T$, that is, p' can be added to it. Hence, the partitioning procedure will evaluate the condition in line 5 as true, and will not abort.

Train p, being a long internal train, does not belong to any chain, i.e., no chain begins or is terminated at s either in $\hat{\mathcal{C}}$ or in $\mathcal{C}^*$. Therefore, $X(\hat{\mathcal{C}}, s+1) = X(\hat{\mathcal{C}}, s)$, $X(\mathcal{C}^*, s+1) = X(\mathcal{C}^*, s)$, and $\Delta X(s+1) = \Delta X(s)$. Moreover, addition of p' increments the total weight of active chains by the same amount in both the sets, and hence does not change the difference, i.e., $\Delta W(s+1) = \Delta W(s)$. Thus, both the invariants hold at $s+1$.

4. *A short external train p', and a hole h or a short internal train p at s*
   By the same argument as in case 3, the partitioning procedure will not abort; rather p' will get added to an active chain in $\hat{\mathcal{C}}$, incrementing the total weight of active chains by the same amount as in $\mathcal{C}^*$. Then, by the argument of case 2, the invariants hold at $s+1$.
5. *A long external train p', and a long internal train p at s*
   By the same argument as in the earlier part of case 3, in $\hat{\mathcal{C}}$ the number of active chains ending upstream of s is more than $6T$. Therefore, the partitioning procedure will evaluate the condition in line 11 as true, and will not abort; rather it will add p' to a chain, say c, which in the current case will be terminated at s. Since one chain is terminated at s out of the $X(\hat{\mathcal{C}}, s)$ chains active on $(s-1, s)$ in $\hat{\mathcal{C}}$, $X(\hat{\mathcal{C}}, s+1) = X(\hat{\mathcal{C}}, s) - 1$. In $\mathcal{C}^*$ too, a chain (the one containing p') is terminated at s, i.e., $X(\mathcal{C}^*, s+1) = X(\mathcal{C}^*, s) - 1$. Hence, Invariant I holds at $s+1$. For Invariant II, we need to consider two subcases :
   a. Suppose in $\hat{\mathcal{C}}$, the chain terminated at s has weight $2T$ or more. Then, by the same argument as in case 2a, Invariant II holds at $s+1$.
   b. Suppose in $\hat{\mathcal{C}}$, the chain terminated at s has weight less than $2T$. Then each of the other upstream ending active chains too must have a weight less than $2T$, and we know that each of the (less than $2T$) downstream ending active chains has weight less than $6T$. The same chains are also active on $(s, s+1)$ with same weights before $s+1$ as before s. Then $X(\hat{\mathcal{C}}, s+1) \leq 9T$ implies $W(\hat{\mathcal{C}}, s+1) < 7T \cdot 2T + 2T \cdot 6T = 26\,T^2$. Since $W(\mathcal{C}^*, s+1) \geq 0$, Invariant II holds at $s+1$.
6. *A long external train p', and a hole h or a short internal train p at s*
   In each of the sets $\hat{\mathcal{C}}$ and $\mathcal{C}^*$, p' is added to a chain which is then terminated at s, and a chain begins at s with h or p. For $\hat{\mathcal{C}}$ by construction, and for $\mathcal{C}^*$ being maximally extended, no other chain is terminated at s. Therefore, $X(\hat{\mathcal{C}}, s+1) = X(\hat{\mathcal{C}}, s)$ and $X(\mathcal{C}^*, s+1) = X(\mathcal{C}^*, s)$, implying that Invariant I holds at $s+1$. For the change in total weights due to the termination of a chain, the same arguments hold as in the cases 5a and 5b. Moreover, the chain that begins at s contributes to the total weight of active chains before $(s+1)$ by the same amount in $\hat{\mathcal{C}}$ as in $\mathcal{C}^*$. Hence, Invariant II also holds at $s+1$. ◀

5.2 Scheduling

If T is a feasible last-entry-time, then a chain set $\hat{\mathcal{C}}$ is returned by the partitioning procedure, and has a congestion $O(T)$ and maximum age $O(T)$. Clearly, the age and congestion are

[5] The last train $p^\dagger$ in such a chain c ends at or downstream of s, and has a path-length less than T since if $p^\dagger$ were long then c would already be terminated before s. Therefore, $p^\dagger$ must originate at one of the $T-1$ nearest stations before s, each of which has at most two trains – one internal and one external.

lower bounds on the last-entry-time with which the chains can be scheduled. However, to schedule them with last-entry-time better than $\Theta(T^2)$, the train-movements and hole-jumps of different chains need to be effectively pipelined. This is the goal of our scheduling procedure.

The procedure first splits the original big problem of scheduling the chains into several small scheduling problems. For this, it assigns a random initial rank $O(T)$ independently to every chain, and then successively incremental ranks to the holes and the movements of non-terminal trains of the chain. Since the total number of train-movements and holes in any chain is $O(T)$, the maximum value of rank assigned is $O(T)$. The original big problem has thus been broken down to as many small problems as the total number $\Gamma = O(T)$ of distinct ranks. The i^{th} small problem consists of the train-movements and holes of rank i, and the goal is to make every hole jump over its entire *extent* (Section 5.2.1) as well as to perform all the train-movements. Each small problem involves at most one hole or train-movement from every chain, and has a congestion $O(\log N)$ w.h.p. Note that the idea of using random delays in order to achieve effective pipelining is not new – it has been used in many previous works, e.g. [14] and [13].

Next, the procedure solves each small problem by using interval graph colouring to partition its set of train-movements and holes into $O(\log N)$ subsets. The colouring ensures that the extents of hole-jumps and train-movements in each subset are mutually disjoint, so that they can be scheduled to take place in a single step.

Thus, the procedure consists of two subroutines – (i) the *ranking subroutine* which assigns the ranks, and (ii) the *scheduling subroutine* which builds the schedule as a sequence of several phases, the i^{th} phase consisting of hole-jumps and train-movements of rank i. The number of distinct values of the ranks is $O(T)$, as we prove in Theorem 10. The number of steps in every phase is $O(\log N)$ w.h.p., as proved in Theorem 11. Hence, the schedule achieves a last-entry-time $O(T \log N)$ w.h.p.

5.2.1 The ranking subroutine

First, to every chain $c \in \hat{\mathcal{C}}$, independently assign a random initial rank $\gamma(c)$ from the range $\{1, 2, \ldots T\}$. Then, to every entry and movement in chain c (except movements following the entry of the terminal train, say $p^\dagger$, of c), assign a rank equal to the sum of $\gamma(c)$ and the number of previous movements and entries in the chain. Let $\gamma(c, \dagger)$ denote the rank of the last entry, that of the terminal train $p^\dagger$, in chain c. Let Γ be the maximum among all ranks.

For every link $(s-1, s)$ on the path of a train p on the line, the *extent of movement* of p across that link is defined as the interval $(s-1, s)$. For every external train p' that enters a station s' filling a hole h either pre-existing or created at some station $s < s'$, the *extent of entry* of p' is defined as (s, s'); the hole h is also said to have the same extent.

5.2.2 The scheduling subroutine

For each $i \in \{1, \ldots, \Gamma\}$, in i^{th} phase, all movements and entries having rank i are scheduled. Since those with overlapping extents cannot be scheduled in same step, a minimal interval colouring is first computed for the set of all extents with rank i. Movements and entries (only those not already scheduled for an earlier step as *incidental movements and entries*) whose extents have colour j are scheduled for the j^{th} step of the i^{th} phase; interval colouring ensures that they do not conflict. Thus, if K_i is the number of colours used in the colouring, then the i^{th} phase has K_i steps. Hence, last-entry-time of the schedule is at most $\sum_{i=1}^{\Gamma} K_i$. All external trains have entered the line by the end of the Γ^{th} phase. Therefore, after that, all trains not already vanished are trivially scheduled to move non-stop to their destinations.

Note that for the j^{th} step of i^{th} phase, along with the *designated* movements and entries (i.e., the ones having rank i and colour j), those movements (with higher value of rank or colour) are also scheduled which do not conflict with the designated ones. Similarly, those non-conflicting entries (with higher rank or colour) also take place in the same step whose holes-to-be-filled happen to move to the respective entry stations in this step. These additional movements and entries are what we earlier qualified as *incidental*.

5.2.3 Analysis

▶ **Theorem 10.** $\Gamma < 7T$

Proof. By definition, rank $\gamma(c, \dagger)$ assigned to the last entry of a chain c is one unit less than the sum of the age of c and its initial rank $\gamma(c) \leq T$. From Theorem 9, the age of every chain in $\hat{\mathcal{C}}$ is less than $6T$. Hence, $\gamma(c, \dagger) < 7T$ for every chain c, and $\Gamma = \max_c \gamma(c, \dagger) < 7T$. ◀

▶ **Theorem 11.** *For each $i \in \{1, \ldots, \Gamma\}$, number of steps in the i^{th} phase is $O(\log N)$ w.h.p.*

Proof. Let $\hat{\mathcal{C}}$ be the set of chains returned by the partitioning procedure for target time T. Let $r \leq \Gamma$ be any rank, and l be any link. From Theorem 9, the congestion produced by $\hat{\mathcal{C}}$ in l is at most $9T$. For each of at most $9T$ chains which cross l, extent of exactly one movement or entry of the chain includes l. Let E_l be the set of all extents which include l. Then $|E_l| \leq 9T$, and the extents in E_l belong to distinct chains.

For every extent $\epsilon \in E_l$, let X_ϵ be a binary random variable which takes a value 1 if the extent ϵ is assigned the rank r, and a value 0 otherwise. Note that the value of X_ϵ depends entirely on the random initial rank $\gamma(c_\epsilon)$ assigned to the chain, say c_ϵ, to which the extent ϵ belongs. Moreover, recall that the random rank is assigned to c_ϵ independently of other chains. Then, since the extents in E_l belong to different chains, X_ϵ are independent Bernoulli random variables. Furthermore, for any extent ϵ, at most one (if any) out of T equally probable values for $\gamma(c_\epsilon)$ can lead to rank r be assigned to ϵ, i.e., $P[X_\epsilon = 1] \leq \frac{1}{T}$.

Let $X := \sum_{\epsilon \in E_l} X_\epsilon$, i.e., X counts the extents which include l and have rank r. Then $\mu := \mathbb{E}[X] \leq \frac{|E_l|}{T} \leq 9$. Applying the Chernoff bound $P[X \geq \lambda] \leq \left(\frac{\lambda}{\mu e}\right)^{-\lambda} \ \forall \lambda \geq 0$, we have :

$P[X \geq k \lg N] \leq \left(\frac{k \lg N}{9e}\right)^{-k \lg N} = \left(\frac{9e}{k \lg N}\right)^{k \lg N} \leq \frac{1}{4^{k \lg N}} = N^{-2k} \ \ \forall k \geq 9e, \forall N \geq 16$

Thus, the probability that l is included in more than $k \lg N$ extents with rank r is less than N^{-2k}. Since there are only N links and only Γ different values for rank, and since $\Gamma < 7T$ from Theorem 10, the probability that any link is included in more than $k \lg N$ extents having same rank is less than $\frac{N \cdot 7T}{N^{2k}} < N^{-k}$. That is, the probability that no more than $k \lg N$ extents of same rank include a common link is greater than $(1 - N^{-k})$. Therefore, with a probability greater than $(1 - N^{-k})$, for every $i \in \{1, \ldots, \Gamma\}$, the number K_i of colours required in the minimal colouring of the extents with rank i is at most $k \lg N$, and hence the number of steps in the i^{th} phase is at most $k \lg N$. ◀

6 Conclusion

The most important open question is whether computing a constant factor approximation for the Train Scheduling problem is NP-hard. Natural generalisations – like multiple platforms at stations, unequal lengths of links, different speeds of trains, relative priorities of trains, and multiple parallel links between stations – should also be studied to bring the model closer to real-life problems.

References

1. V. Cacchiani, D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, and J. Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014. doi:10.1016/j.trb.2014.01.009.
2. X. Cai, C.J. Goh, and Alistair I. Mees. Greedy heuristics for rapid scheduling of trains on a single track. *IIE Transactions*, 30(5):481–493, May 1998. doi:10.1023/A:1007551424010.
3. Gabrio Caimi, Fabián A. Chudak, Martin Fuchsberger, Marco Laumanns, and Rico Zenklusen. A new resource-constrained multicommodity flow model for conflict-free train routing and scheduling. *Transportation Science*, 45(2):212–227, 2011. doi:10.1287/trsc.1100.0349.
4. Te-Wei Chiang, Hai-Yen Hau, Hwan-Ming Chiang, Su-Yun Ko, and Chao-Ho Hsieh. Knowledge-based system for railway scheduling. *Data Knowl. Eng.*, 27(3):289–312, 1998. doi:10.1016/S0169-023X(97)00040-2.
5. Andrea E. F. Clementi and Miriam Di Ianni. Optimum schedule problems in store and forward networks. In *Proceedings IEEE INFOCOM '94, The Conference on Computer Communications, Thirteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Networking for Global Communications, Toronto, Ontario, Canada, June 12-16, 1994*, pages 1336–1343. IEEE, 1994. doi:10.1109/INFCOM.1994.337560.
6. Andrea D'Ariano. *Improving real-time train dispatching: Models, algorithms and applications*. Doctoral thesis, TRAIL Research School, Deft, The Netherlands, 2008.
7. Wei Fang, Shengxiang Yang, and Xin Yao. A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Trans. Intelligent Transportation Systems*, 16(6):2997–3016, 2015. doi:10.1109/TITS.2015.2446985.
8. Holger Flier, Matús Mihalák, Anita Schöbel, Peter Widmayer, and Anna Zych. Vertex disjoint paths for dispatching in railways. In Thomas Erlebach and Marco E. Lübbecke, editors, *ATMOS 2010 - 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Liverpool, United Kingdom, September 6-10, 2010*, volume 14 of *OASICS*, pages 61–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010. doi:10.4230/OASIcs.ATMOS.2010.61.
9. 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.
10. J. Harbering, A. Ranade, and M. Schmidt. Single track train scheduling. In Z. Hanzalek, G. Kendall, B. McCollum, and P. Sucha, editors, *In proceedings of the 7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015), 25 - 28 Aug 2015, Prague, Czech Republic*, pages 102–117, 2015.
11. R. V. Iyer and S. Ghosh. Daryn-a distributed decision-making algorithm for railway networks: modeling and simulation. *IEEE Transactions on Vehicular Technology*, 44(1):180–191, February 1995. doi:10.1109/25.350284.
12. Johanna Törnquist Krasemann. Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances. *Transportation Research Part C: Emerging Technologies*, 20(1):62–78, 2012. Special issue on Optimization in Public Transport+ISTT2011. doi:10.1016/j.trc.2010.12.004.
13. Frank Thomson Leighton, Bruce M. Maggs, Abhiram G. Ranade, and Satish Rao. Randomized routing and sorting on fixed-connection networks. *J. Algorithms*, 17(1):157–205, 1994. doi:10.1006/jagm.1994.1030.
14. Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994. doi:10.1007/BF01215349.

15 Frank Thomson Leighton, Bruce M. Maggs, and Andréa W. Richa. Fast algorithms for finding o(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999. doi:10.1007/s004930050061.

16 Joseph Y.-T. Leung, Tommy W. Tam, and Gilbert H. Young. On-line routing of real-time messages. *J. Parallel Distrib. Comput.*, 34(2):211–217, 1996. doi:10.1006/jpdc.1996.0057.

17 Carlo Mannino and Alessandro Mascis. Optimal real-time traffic control in metro stations. *Operations Research*, 57(4):1026–1039, 2009. doi:10.1287/opre.1080.0642.

18 Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002. doi:10.1016/S0377-2217(01)00338-1.

19 Sundaravalli Narayanaswami and Narayan Rangaraj. Scheduling and rescheduling of railway operations: A review and expository analysis. *Technology Operation Management*, 2(2):102–122, December 2011. doi:10.1007/s13727-012-0006-x.

20 Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In Evripidis Bampis and Klaus Jansen, editors, *Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers*, volume 5893 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 2009. doi:10.1007/978-3-642-12450-1_20.

21 Paola Pellegrini and Joaquin Rodriguez. Single european sky and single european railway area: A system level analysis of air and rail transportation. *Transportation Research Part A: Policy and Practice*, 57:64–86, 2013. doi:10.1016/j.tra.2013.09.004.

22 Thomas Rothvoß. A simpler proof for o(congestion + dilation) packet routing. *CoRR*, abs/1206.3718, 2012. arXiv:1206.3718.

23 Ismail Sahin. Railway traffic control and train scheduling based oninter-train conflict management. *Transportation Research Part B: Methodological*, 33(7):511–534, September 1999. doi:10.1016/S0191-2615(99)00004-1.

24 Christian Scheideler. Universal routing strategies for interconnection networks. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Lecture Notes in Computer Science 1390*, pages 57–67. W. H. Freeman & Co., New York, NY, USA, 1998.

On the Control of Asynchronous Automata*†

Hugo Gimbert

LaBRI, CNRS, Université de Bordeaux, France
hugo.gimbert@cnrs.fr

Abstract

The decidability of the distributed version of the Ramadge and Wonham controller synthesis problem [11], where both the plant and the controllers are modeled as asynchronous automata [12, 1] and the controllers have causal memory is a challenging open problem [9, 7]. There exist three classes of plants for which the existence of a correct controller with causal memory has been shown decidable: when the dependency graph of actions is series-parallel, when the processes are connectedly communicating and when the dependency graph of processes is a tree. We design a class of plants, called decomposable games, with a decidable controller synthesis problem. This provides a unified proof of the three existing decidability results as well as new examples of decidable plants.

1998 ACM Subject Classification B.1.2 Automatic synthesis, H.3.4 Distributed systems

Keywords and phrases Asynchronous automata, Controller synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.30

1 Introduction

The decidability of the distributed version of the Ramadge and Wonham control problem [11], where both the plant and the controllers are modeled as asynchronous automata [12, 1] and the controllers have causal memory is a challenging open problem. Very good introductions to this problem are given in [9, 7].

In this setting a controllable plant is distributed on several finite-state processes which interact asynchronously using shared actions. On every process, the local controller can choose to block some of the actions, called *controllable* actions, but it cannot block the *uncontrollable* actions from the environment. The choices of the local controllers are based on two sources of information.

- First the controller monitors the sequence of states and actions of the local process. This information is called the *local view* of the controller.
- Second when a shared action is played by several processes then all the controllers of these processes can exchange as much information as they want. In particular together they can compute their mutual view of the global execution: their *causal past*.

A controller is correct if it guarantees that every possible execution of the plant satisfies some specification. The controller synthesis problem is a decision problem which, given a plant as input, asks whether the system admits a correct controller. In case such a controller exists, the algorithm should compute one as well.

The difficulty of controller synthesis depends on several factors, e.g.:

- the size and architecture (pipeline, ring, ...) of the system,

* A full version of this paper, including proofs, is available as a technical report [5].
† We acknowledge support from ANR-13-BS02-0011 "Stoch-MC" and UMI Relax.

- the information available to the controllers,
- the specification.

Assuming that processes can exchange information upon synchronization and use their causal past to take decisions is one of the key aspects to get decidable synthesis problems [3]. In early work on distributed controller synthesis, for example in the setting of [10], the only source of information available to the controllers is their local view. In this setting, distributed synthesis is not decidable in general, except for very particular architectures like the pipeline architecture. The paper [2] proposes information forks as a uniform notion explaining the (un)decidability results in distributed synthesis. The idea of using causal past as a second source of information appeared in [3].

We adopt a modern terminology and call the plant a *distributed game* and the controllers are *distributed strategies* in this game. A distributed strategy is a function that maps the causal past of processes to a subset of controllable actions. In the present paper we focus on the *termination condition*, which is satisfied when each process is guaranteed to terminate its computation in finite time, in a final state. A distributed strategy is winning if it guarantees the termination condition, whatever uncontrollable actions are chosen by the environment.

We are interested in the following problem, whose decidability is an open question.

DISTRIBUTED SYNTHESIS PROBLEM: given a distributed game decide whether there exists a winning strategy.

There exist three classes of plants for which the DISTRIBUTED SYNTHESIS PROBLEM has been shown decidable:

1. when the dependency graph of actions is series-parallel [3],
2. when the processes are connectedly communicating [6],
3. and when the dependency graph of processes is a tree [4, 8].

A series-parallel game is a game such that the dependency graph of the alphabet A is a co-graph. Series-parallel games were proved decidable in [3], for a different setup than ours: in the present paper we focus on process-based control while [3] was focusing on action-based control. Actually action-based control is more general than process-based control, see [9] for more details. The results of the present paper could probably be extended to action-based control however we prefer to stick to process-based control in order to keep the model intuitive. To our knowledge, the result of [3] was the first discovery of a class of asynchronous distributed system with causal memory for which the DISTRIBUTED SYNTHESIS PROBLEM is decidable.

Connectedly communicating games have been introduced [6]. A game is connectedly communicating if there is a bound k such that if a process p executes k steps in parallel to another process q then all further actions of p will be parallel to q. The event structure of a connectedly communicating games has a decidable MSO theory [6] which implies that the DISTRIBUTED SYNTHESIS PROBLEM is decidable for these games.

An acyclic game is a game where processes are arranged as a tree and actions are either local or synchronize a father and its son. Even in this simple setting the DISTRIBUTED SYNTHESIS PROBLEM is non-elementary hard [4].

Our contribution

We develop a new proof technique to address the DISTRIBUTED SYNTHESIS PROBLEM, and provide a unified proof of decidability for series-paralell, connectedly communicating and acyclic games. We design a class of games, called *decomposable games*, for which the

DISTRIBUTED SYNTHESIS PROBLEM is decidable. This leads to new examples of decidable architectures for controller synthesis.

The winning condition of the present paper is the termination of all processes in a final state. Richer specifications can be expressed by parity conditions. In the present paper we stick to termination conditions for two reasons. First, the long-term goal of this research is to establish the decidability or undecidability of the distributed controller synthesis problem. A possible first step is to prove decidability for games with termination conditions. Second, it seems that the results of the present paper can be lifted to parity games, using the same concepts but at the cost of some extra technical details needed to reason about infinite plays.

Our proof technique consists in simplifying a winning strategy by looking for useless parts to be removed in order to get a smaller winning strategy. These parts are called *useless repetitions*. Whenever a useless repetition exists, we remove it using an operation called a *shortcut* in order to get a simpler strategy. Intuitively, a shortcut is a kind of cut-and-paste operation which makes the strategy smaller. By taking shortcuts again and again, we make the strategy smaller and smaller, until it does not have any useless repetition anymore.

If a winning strategy exists, there exists one with no useless repetition. In decomposable games, there is a computable upper bound on the size of strategies with no useless repetition, which leads to decidability of the controller synthesis problem.

Performing cut-and-paste in a distributed game is not as easy as doing it in a single-process game. In a single-process game, strategies are trees and one can cut a subtree from a node A and paste it to any other node B, and the operation makes sense as long as the state of the process is the same in both nodes. In the case of a general distributed strategy, designing cut-and-paste operations is more challenging. Such operations on the strategy tree should be consistent with the level of information of each process, in order to preserve the fundamental property of distributed strategies: the decisions taken by a process should depend only on its causal view, not on parallel events.

The decidability of series-parallel games established in [3] relies also on some simplification of the winning strategies, in order to get *uniform* strategies. The series-parallel assumption is used to guarantee that the result of the replacement of a part of a strategy by a uniform strategy is still a strategy, as long as the states of all processes coincide. Here we work without the series-parallel assumption, and matching the states is not sufficient for a cut-and-paste operation to be correct.

This is the reason for introducing the notion of *lock*. A lock is a part of a strategy where information is guaranteed to spread in a team of processes before any of these processes synchronize with a process outside the team. When two locks A and B are similar, in some sense made precise in the paper, the lock B can be cut and paste on lock A. Upon arrival on A, a process of the team initiates a change of strategy, which progressively spreads across the team. All processes of the team should eventually play as if the play from A to B had already taken place, although it actually did not.

The complexity of our algorithm is really bad, so probably this work has no immediate practical applications. This is not surprising since the problem is non-elementary even for the class of acyclic games [4]. Nevertheless we think this paper sheds new light on the difficult open problem of distributed synthesis.

Organization of the paper

Section 2 introduces the DISTRIBUTED SYNTHESIS PROBLEM. Section 3 provides several examples. In section 4 we show how to simplify strategies which contain useless repetitions, and prove that if a winning strategy exists, there exists one without any useless repetition.

Finally, section 5 introduces the class of decomposable games and show their controller synthesis problem is decidable. A full version of this paper, including proofs, is available as a technical report [5].

2 The distributed synthesis problem

The theory of Mazurkiewicz traces is very rich, for a thorough presentation see [1]. Here we only fix notations and recall the notions of traces, views, prime traces and parallel traces.

We fix an alphabet A and a symmetric and reflexive dependency relation $D \subseteq A \times A$ and the corresponding independency relation $\mathbb{I} \subseteq A \times A$ defined as $\forall a, b \in A, (a \mathbb{I} b) \iff (a, b) \notin D$. A *Mazurkiewicz trace* or, more simply, a *trace*, is an equivalence class for the smallest equivalence relation $\equiv$ on A^* which commutes independent letters i.e. for every letters a, b and every words w_1, w_2,

$$a \mathbb{I} b \implies w_1abw_2 \equiv w_1baw_2 \ .$$

The words in the equivalence class are the *linearizations* of the trace. The trace whose only linearization is the empty word is denoted ϵ. All linearizations of a trace u have the same set of letters and length, denoted respectively $\mathrm{Alph}(u)$ and $|u|$. Given $B \subseteq A$, the set of traces such that $\mathrm{Alph}(u) \subseteq B$ is denoted $B^*_{\equiv}$ in particular the set of all traces is $A^*_{\equiv}$.

The concatenation on words naturally extends to traces. Given two traces $u, v \in A^*_{\equiv}$, the trace uv is the equivalence class of any word in uv. The prefix relation $\sqsubseteq$ is defined by $(u \sqsubseteq v \iff \exists w \in A^*_{\equiv}, uw = v)$. and the suffix relation is defined similarly.

Maxima, prime traces and parallel traces

A letter $a \in A$ is a *maximum* of a trace u if it is the last letter of one of the linearizations of u. A trace $u \in A^*_{\equiv}$ is *prime* if it has a unique maximum, denoted $\mathrm{last}(u)$ and called the last letter of u. Two prime traces u and v are said to be *parallel* if

- neither u is a prefix of v nor v is a prefix of u; and
- there is a trace w such that both u and v are prefixes of w. These notions are illustrated on Fig. 1.

Processes and automata

Asynchronous automata are to traces what finite automata are to finite words, as witnessed by Zielonka's theorem [12]. An asynchronous automaton is a collection of automata on finite words, whose transition tables do synchronize on certain actions.

▶ **Definition 1.** An asynchronous automaton on alphabet A with processes $\mathbb{P}$ is a tuple $\mathcal{A} = ((A_p)_{p\in\mathbb{P}}, (Q_p)_{p\in\mathbb{P}}, (i_p)_{p\in\mathbb{P}}, (F_p)_{p\in\mathbb{P}}, \Delta)$ where:

- every process $p \in \mathbb{P}$ has a set of actions A_p, a set of states Q_p and $i_p \in Q_p$ is the initial state of p and $F_p \subseteq Q_p$ its set of final states.
- $A = \bigcup_{p\in\mathbb{P}} A_p$. For every letter $a \in A$, the domain of a is $\mathrm{dom}(a) = \{p \in \mathbb{P} \mid a \in A_p\}$.
- Δ is a set of transitions of the form $(a, (q_p, q'_p)_{p\in\mathrm{dom}(a)})$ where $a \in A$ and $q_p, q'_p \in Q_p$. Transitions are *deterministic*: for every $a \in A$, if $\delta = (a, (q_p, q'_p)_{p\in\mathrm{dom}(a)}) \in \Delta$ and $\delta' = (a, (q_p, q''_p)_{p\in\mathrm{dom}(a)}) \in \Delta$ then $\delta = \delta'$ (hence $\forall p \in \mathrm{dom}(a), q'_p = q''_p$).

Such an automaton works asynchronously: each time a letter a is processed, the states of the processes in $\mathrm{dom}(a)$ are updated according to the corresponding transition, while the

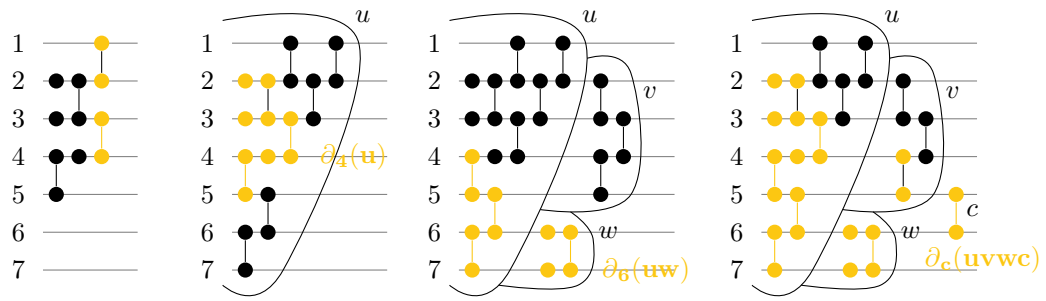


Figure 1 The set processes is $\{1 \dots 7\}$. A letter is identified with its domain. Here the domains are either singletons, represented by a single dot, or pairs of contigous processes, represented by two dots connected with a vertical segment. The trace $\{2\}\{3\}\{4,5\}\{2,3\}\{4\}\{1,2\}\{3,4\} = \{4,5\}\{4\}\{2\}\{3\}\{2,3\}\{3,4\}\{1,2\}$ is represented on the left-handside. It has two maximal letters $\{1,2\}$ and $\{3,4\}$ thus is not prime. Center left: process 4 sees only its causal view $\partial_4(u)$ (in yellow). Center right: $uvw = uwv$ since $\mathrm{dom}(v) \cap \mathrm{dom}(w) = \emptyset$. Both uv and $\partial_6(uw)$ (in yellow) are prime prefixes of uvw and they are parallel. Right: uv and $\partial_c(uvwc)$ (in yellow) are parallel.

states of other processes do not change. This induces a natural commutation relation $\mathbb{I}$ on A: two letters commute iff they have no process in common i.e.

$$(a \mathbb{I} b) \iff (\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset) \ .$$

The set of *plays* of the automaton $\mathcal{A}$ is a set of traces denoted plays($\mathcal{A}$) and defined inductively, along with a mapping state : plays($\mathcal{A}$) $\to \Pi_{p\in\mathbb{P}} Q_p$.

- ϵ is a play and $\mathrm{state}(\epsilon) = (i_p)_{p\in\mathbb{P}}$,
- for every play u such that $(\mathrm{state}_p(u))_{p\in\mathbb{P}}$ is defined and $\big(a, (\mathrm{state}_p(u), q_p)_{p\in\mathrm{dom}(a)}\big)$ is a transition then ua is a play and $\forall p \in \mathbb{P}, \mathrm{state}_p(ua) = \begin{cases} \mathrm{state}_p(u) & \text{if } p \notin \mathrm{dom}(a), \\ q_p & \text{otherwise.} \end{cases}$

For every play u, state(u) is called the *global state* of u. The inductive definition of state(u) is correct because it is invariant by commutation of independent letters of u.

Counting actions of a process

For every trace u we can count how many times a process p has played an action in u, which we denote $|u|_p$. Formally, $|u|_p$ is first defined for words, as the length of the projection of u on A_p, which is invariant by commuting letters. The domain of a trace is defined as

$$\mathrm{dom}(u) = \{p \in \mathbb{P} \mid |u|_p \neq 0\} \ .$$

Views, strategies and games

Given an automaton $\mathcal{A}$, we want the processes to choose actions which guarantee that every play eventually terminates in a final state.

To take into account the fact that some actions are controllable by processes while some other actions are not, we assume that A is partitioned in

$$A = A_c \sqcup A_e$$

where A_c is the set of controllable actions and A_e the set of (uncontrollable) environment actions. Intuitively, processes cannot prevent their environment to play actions in A_e, while they can decide whether to block or allow any action in A_c.

We adopt a modern terminology and call the automaton $\mathcal{A}$ together with the partition $A = A_c \sqcup A_e$ a *distributed game*, or even more simply a *game*. In this game the processes play distributed strategies, which are individual plans of action for each process. The choice of actions by a process p is dynamic: at every step, p chooses a new set of controllable actions, depending on its information about the way the play is going on. This information is limited since processes cannot communicate together unless they synchronize on a common action. In that case however they exchange as much information about the play as they want. Finally, the information missing to a process is the set of actions which happened in parallel of its own actions. The information which remains is called the p-view of the play, it is illustrated on Fig. 1 and defined formally as follows.

▶ **Definition 2** (Views)**.** For every set of processes $\mathbb{Q} \subseteq \mathbb{P}$ and trace u, the $\mathbb{Q}$-*view* of u, denoted $\partial_{\mathbb{Q}}(u)$, is the unique trace such that u factorizes as $u = \partial_{\mathbb{Q}}(u) \cdot v$ and v is the longest suffix of u such that $\mathbb{Q} \cap \mathrm{dom}(v) = \emptyset$. In case $\mathbb{Q}$ is a singleton $\{p\}$ the view is denoted $\partial_p(u)$ and is either empty or prime. For every letter $a \in A$ we denote $\partial_a(u) = \partial_{\mathrm{dom}(a)}(u)$.

Some useful properties of the $\mathbb{Q}$-view are:

$$\partial_{\mathbb{Q}}(uv) = \partial_{\mathbb{Q}'}(u)\,\partial_{\mathbb{Q}}(v) \text{ where } \mathbb{Q}' = \mathbb{Q} \cup \mathrm{dom}(\partial_{\mathbb{Q}}(v)) \tag{1}$$

$$(\mathbb{Q} \subseteq \mathbb{Q}') \implies (\partial_{\mathbb{Q}}(u) \sqsubseteq \partial_{\mathbb{Q}'}(u)) \ . \tag{2}$$

We can now define what is a distributed strategy.

▶ **Definition 3** (Distributed strategies, consistent and maximal plays)**.** Let $G = (\mathcal{A}, A_c, A_e)$ be a distributed game. A *strategy for process* p in G is a mapping which associates with every play u a set of actions $\sigma_p(u)$ such that:

- environment actions are allowed: $A_e \subseteq \sigma_p(u)$,
- the decision depends only on the view of the process: $\sigma_p(u) = \sigma_p(\partial_p(u))$.

A *distributed strategy* is a tuple $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ where each σ_p is a strategy of process p. A play $u = a_1 \cdots a_{|u|} \in \mathrm{plays}(\mathcal{A})$ is *consistent with* σ, or equivalently is a σ-*play* if:

$$\forall i \in 1 \ldots |u|, \forall p \in \mathrm{dom}(a_i), a_i \in \sigma_p(a_1 \cdots a_{i-1}) \ .$$

A σ-play is *maximal* if it is not the strict prefix of another σ-play.

Note that a strategy is forced to allow every environment action to be executed at every moment. This may seem to be a huge strategic advantage for the environment. However depending on the current state, not every action can be effectively used in a transition because the transition function is not assumed to be total. So in general not every environment actions can actually occur in a play. In particular it may happen that a process enters a final state with no outgoing transition, where no uncontrollable action can happen.

Winning games

Our goal is to synthesize strategies which ensure that the game terminates and all processes are in a final state.

▶ **Definition 4** (Winning strategy)**.** A strategy σ is winning if the set of σ-plays is finite and in every maximal σ-play u, every process is in a final state i.e. $\forall p \in \mathbb{P}, \mathrm{state}_p(u) \in F_p$.

We are interested in the following problem, whose decidability is an open question.

DISTRIBUTED SYNTHESIS PROBLEM: given a distributed game decide whether there exists a winning strategy.

If the answer is positive, the algorithm should compute a winning strategy as well.

3 Three decidable classes

Series-parallel games. A game is *series-parallel* if its dependency alphabet (A, D) is a co-graph i.e. belongs to the smallest class of graphs containing singletons and closed under parallel product and complementation. In this case A has a *decomposition tree*, this is a binary tree whose nodes are subsets of A, its leaves are the singletons $(\{a\})_{a\in A}$, its root is A. Moreover every node B with two children B_0 and B_1 is the disjoint union of B_0 and B_1 and either $B_0 \times B_1 \subseteq D$ (serial product) or $(B_0 \times B_1) \cap D = \emptyset$ (parallel product).

The synthesis problem is decidable for series-parallel games [3].

Connectedly communicating games. A game is *k-connectedly communicating* if for every pair p, q of processes, if process p plays k times in parallel of process q then all further actions of q will be parallel to p. Formally, for every prime play uvw, $(q \notin \mathrm{dom}(v)$ and $|v|_p \geq k) \implies q \notin \mathrm{dom}(w)$.

The MSO theory of the event structure of a k-connectedly communicating game is decidable [6], which implies that controller synthesis is decidable for theses games.

Acyclic games. An acyclic game is a game where processes $\mathbb{P}$ are the nodes of a tree $T_{\mathbb{P}}$ and the domain of every action is a connected set of nodes of $T_{\mathbb{P}}$. The synthesis problem is known to be decidable for acyclic games such that the domain of each action has size 1 or 2 [4].

4 Simplifying strategies

In this section we present an elementary operation called a *shortcut*, which can be used to simplify and reduce the duration of a winning strategy.

To create a shortcut, one selects a σ-play xy and modifies the strategy σ so that as soon as any of the processes sees the play x in its view, this process assumes that not only x but also xy has actually occurred. In other words, a shortcut is a kind of *cut-and-paste* in the strategy: we glue on node x the sub-strategy rooted at node xy.

The choice of x and y should be carefully performed so that the result of the shortcut is still a strategy. We provide a sufficient condition for that: (x, y) should be a *useless repetition*.

The interest of taking shortcuts is the following: if the original strategy is winning, then the strategy obtained by taking the shortcut is winning as well, and strictly smaller than the original one. In the remainder of this section, we formalize these concepts.

4.1 Locks

We need to limit the communication between a set of processes, called a team, and processes outside the team. This leads to the notion of a $\mathbb{Q}$-lock: this is a prime play u such that there is no synchronization between $\mathbb{Q}$ and $\mathbb{P} \setminus \mathbb{Q}$ in parallel of u.

▶ **Definition 5.** Let $\mathbb{Q} \subseteq \mathbb{P}$. An action b is $\mathbb{Q}$-safe if ($\mathrm{dom}(b) \subseteq \mathbb{Q}$ or $\mathrm{dom}(b) \cap \mathbb{Q} = \emptyset$). A play u is a $\mathbb{Q}$-*lock* if it is prime and the last action of every prime play parallel to u is $\mathbb{Q}$-safe.

The notion of lock is illustrated on the right handside of Fig. 1. Set $\mathbb{Q} = \{1, 2, 3, 4, 5\}$. Then uv is not a $\mathbb{Q}$-lock because $\partial_c(uvwc)$ is parallel to uv but c is not $\mathbb{Q}$-safe. Locks occur in a variety of situations, including the three decidable classes.

▶ **Lemma 6** (Sufficient conditions for $\mathbb{Q}$-locks)**.** *Let u be a prime play of a game G and $\mathbb{Q} \subseteq \mathbb{P}$. Each of the following conditions is sufficient for u to be a $\mathbb{Q}$-lock:*

(i) $\mathbb{Q} = \mathbb{P}$.
(ii) *u is a $(\mathbb{P} \setminus \mathbb{Q})$-lock.*
(iii) $\mathbb{Q} \subseteq \mathrm{dom}(\mathrm{last}(u))$.
(iv) *The game is series-parallel and $\mathbb{Q} = \mathrm{dom}(B)$ where B is the smallest node of the decomposition tree of A which contains* $\mathrm{Alph}(u)$.
(v) *The game is connectedly communicating game with bound k, $\mathbb{Q} = \mathrm{dom}(u)$ and $\forall p \in dom(u), |u|_p \geq k$.*
(vi) *The game is acyclic with respect to a tree $T_{\mathbb{P}}$ and $\mathbb{Q}$ is the set of descendants in $T_{\mathbb{P}}$ of the processes in* $\mathrm{dom}(\mathrm{last}(u))$.
(vii) *There are two traces x and z such that $u = xz$ and z is a $\mathbb{Q}$-lock in the game G_x identical to G except the initial state is changed to* $\mathrm{state}(x)$.

4.2 Taking shortcuts

In this section we present a basic operation used to simplify a strategy, called a *shortcut*, which consists in modifying certain parts of a strategy, called *useless repetitions*. These notions rely on the notion of *strategic state* as well as two operations on strategies called *shifting* and *projection*.

▶ **Definition 7** (Residual)**.** Let σ be a strategy, u a σ-play and $\mathbb{Q} \subseteq \mathbb{P}$. The $\mathbb{Q}$-residual of σ after u is the set:

$$\pi(\sigma, u, \mathbb{Q}) = \{(v, \sigma(uv)) \mid v \in A^*_{\equiv}, \mathrm{dom}(v) \subseteq \mathbb{Q} \text{ and } uv \text{ is a } \sigma\text{-play.}\} \ .$$

A winning strategy may take unnecessarily complicated detours in order to ensure termination. Such detours are called *useless repetitions*.

▶ **Definition 8** (Strategic state)**.** Let $\mathbb{Q} \subseteq \mathbb{P}$ be a set of processes, σ a strategy and u a prime σ-play with maximal letter b. The strategic $\mathbb{Q}$-state of σ after u is the tuple

$$\mathrm{strate}_{\sigma,\mathbb{Q}}(u) = (b, \mathrm{state}(u), \pi\left(\sigma, u, \mathbb{Q} \setminus \mathrm{dom}(b)\right)) \ .$$

▶ **Definition 9** (Useless repetition)**.** A useless $\mathbb{Q}$-repetition in a strategy σ is a pair of traces (x, y) such that y is not empty, xy is a σ-play, $\mathrm{dom}(y) \subseteq \mathbb{Q}$, both x and xy are $\mathbb{Q}$-locks and $\mathrm{strate}_{\sigma,\mathbb{Q}}(x) = \mathrm{strate}_{\sigma,\mathbb{Q}}(xy)$.

The following theorem is the key to our decidability results.

▶ **Theorem 10.** *If there exists a winning strategy then there exists a winning strategy without any useless repetition.*

The proof of this theorem relies on the notion of shortcuts, an operation which turns a winning strategy into another strategy with strictly shorter duration.

▶ **Definition 11** (Duration of a strategy). The duration of a strategy σ is

$$\mathrm{dur}(\sigma) = \sum_{u \text{ maximal } \sigma\text{-play}} |u| \ .$$

The duration of a strategy σ may in general be infinite but is finite if σ is winning.

▶ **Lemma 12.** *Let (x, y) be a useless $\mathbb{Q}$-repetition in a strategy σ. Let $\Phi : A^*_{\equiv} \to A^*_{\equiv}$ and τ defined by* $\Phi(u) = \begin{cases} u \text{ if } x \not\sqsubseteq u \\ xyu' \text{ if } x \sqsubseteq u \text{ and } u = xu' \end{cases}$ *and*

$\forall p \in \mathbb{P}, \tau_p(u) = \sigma_p(\Phi(\partial_p(u)))$.

1. *Then τ is a strategy called the (x, y)*-shortcut of σ. *Moreover for every trace u,*

$$(u \text{ is a } \tau\text{-play}) \iff (\Phi(u) \text{ is a } \sigma\text{-play}) \ . \tag{3}$$

2. *If σ is a winning strategy then τ is winning as well and has a strictly smaller duration.*

The strategy τ can be summarized as asking to every process "(†) whenever play x has occurred, replace it by xy and apply σ". Claim 1 in Lemma 12 would not hold in general if (x, y) would not be a useless repetition. For example, assume $u = x$ in the definition above. In general, right after x has occurred, a process p which is not part of the domain of the maximal action of x is playing in parallel of x and sees a strict prefix $\partial_p(x)$ of x. Thus p does not know whether the play x has actually occurred. Asking this process to apply (†) is "cheating" because by definition of strategies, the decision of p after play x should be based only on $\partial_p(x)$. However, if (x, y) is a useless repetition then τ is a distributed strategy, which relies on the equality $\sigma_p(\Phi(\partial_p(u))) = \sigma_p(\partial_p(\Phi(u)))$ for every play u.

Claim 2 relies on $\mathrm{dur}(\tau) < \mathrm{dur}(\sigma)$ which follows immediately from (3). And according to (3) again, the set of global states of the maximal plays is the same for σ and τ thus if σ is winning then τ is winning as well.

Proof of Theorem 10. As long as there exists a useless repetition, take the corresponding shortcut. According to Lemma 12, this creates a sequence $\sigma_0, \sigma_1, \ldots$ of winning strategies whose duration strictly decreases. Thus the sequence is finite and its last element is a winning strategy without useless repetition. ◀

5 Decomposable games

In this section we introduce *decomposable* games, for which the DISTRIBUTED SYNTHESIS PROBLEM is decidable (Theorem 21). There are actually three notions of decomposability: structural decomposability, process decomposability and action decomposability. These three notions form a hierarchy: structural decomposability implies process decomposability which itself implies action decomposability (Lemma 19). Known decidable classes are decomposable: acyclic games are structurally decomposable (Lemma 14), connectedly-communicating games are process decomposable (Lemma 16) and series-parallel games are action decomposable (Lemma 18). Structural decomposability is stable under some operations between games which leads to new examples of decidable games (Lemma 26).

5.1 Decomposability

The notions of decomposability rely on *preorders* defined on $2^{\mathbb{P}}$ or 2^A. A preorder $\preceq$ is a reflexive and transitive relation. We denote $\prec$ the relation $(x \prec y) \iff (x \preceq y \wedge y \not\preceq x)$.

Structural decomposability. This notion of decomposability relies on a preorder $\preceq$ on $2^{\mathbb{P}}$ which is monotonic with respect to inclusion, i.e. $\forall \mathbb{Q}, \mathbb{Q}' \subseteq \mathbb{P}$, $(\mathbb{Q} \subseteq \mathbb{Q}' \implies \mathbb{Q} \preceq \mathbb{Q}')$.

▶ **Definition 13** (Structural decomposability)**.** A game is $\preceq$-*structurally decomposable* if for every non-empty prime trace $y \in A^*_{\equiv}$ there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and $b \in \mathrm{Alph}(y)$ such that:

$$(\mathbb{Q} \setminus \mathrm{dom}(b)) \prec \mathbb{Q}$$
$$\forall a \in A, (a \mathbb{I} b \implies a \text{ is } \mathbb{Q}\text{-safe}) \ .$$

We say a game is structurally decomposable if it is $\preceq$-structurally decomposable for some preorder $\preceq$. We have already seen one example of such games.

▶ **Lemma 14.** *Acyclic games are structurally decomposable.*

Proof. Assume the game is acyclic with process tree $T_{\mathbb{P}}$. Set $\mathbb{Q} \preceq \mathbb{Q}'$ iff every process in $\mathbb{Q}$ has a $T_{\mathbb{P}}$-ancestor in $\mathbb{Q}'$, which is monotonic with respect to inclusion. Let y be a prime trace, $p \in \mathbb{P}$ the least common ancestor in $T_{\mathbb{P}}$ of processes in $\mathrm{dom}(y)$ and $\mathbb{Q}$ the set of descendants of p. Then $\mathrm{dom}(y) \subseteq \mathbb{Q}$. Moreover, since y is prime and since the domain of every action is a connected subset of $T_{\mathbb{P}}$ then $\mathrm{dom}(y)$ is connected as well thus $p \in \mathrm{dom}(y)$ and there exists a letter $b \in \mathrm{Alph}(y)$ such that $p \in \mathrm{dom}(b)$. We show that b satisfies the conditions in the definition of structural decomposability. First, $(\mathbb{Q} \setminus \mathrm{dom}(b)) \preceq \mathbb{Q}$ and the inequality is strict because the only ancestor of p in $\mathbb{Q}$ is p itself and $p \in \mathrm{dom}(b)$. Second, let $a \in A$ such that $a \mathbb{I} b$. Then $p \notin \mathrm{dom}(a)$ and since $\mathrm{dom}(a)$ is connected in $T_{\mathbb{P}}$, then either none of the processes in $\mathrm{dom}(a)$ or all of them are descendants of p in $T_{\mathbb{P}}$, i.e. a is $\mathbb{Q}$-safe. ◀

Process decomposability. The definition of process decomposable games relies on a parameter $k \in \mathbb{N}$ and a preorder $\preceq$ on $2^{\mathbb{P}}$ which is monotonic with respect to inclusion.

▶ **Definition 15** (Process decomposable games)**.** Fix an integer k. A trace y is k-*repeating* if

$$y \text{ is not empty and } \forall p \in \mathrm{dom}(y), |y|_p \geq k \ .$$

A game is $(\preceq, k)$-*process decomposable* if for every prime play xy, if y is k-repeating then there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and a prime prefix $z \sqsubseteq y$ such that $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock and

$$(\mathbb{Q} \setminus \mathrm{dom}(\mathrm{last}(z))) \prec \mathrm{dom}(y) \ . \tag{4}$$

We have already seen one example of process decomposable games.

▶ **Lemma 16.** *Connectedly communicating games are process decomposable.*

Action decomposability. Action decomposability is defined with respect to a parameter $k \in \mathbb{N}$ and a preorder $\preceq$ on 2^A which is monotonic with respect to inclusion.

▶ **Definition 17** (Action decomposable games)**.** Let k be an integer. A game is $(\preceq, k)$ *action decomposable* if for every prime play xy such that y is k-repeating, there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and a prime prefix $z \sqsubseteq y$ such that $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock and

$$\{a \in A \mid \mathrm{dom}(a) \subseteq (\mathbb{Q} \setminus \mathrm{dom}(\mathrm{last}(z))\} \prec \mathrm{Alph}(y) \ .$$

We have already seen one example of action decomposable games.

▶ **Lemma 18.** *Series-parallel games are action decomposable.*

Finally we show that these notions form a hierarchy.

▶ **Lemma 19.** *Every structurally decomposable game is process decomposable and every process decomposable game is action decomposable.*

Thus *action decomposability* is the most general notion of decomposability. In the sequel for the sake of conciseness, it is simply called *decomposability*.

5.2 Decidability

In this section we show that decomposability is a decidable property and decomposable games have a decidable controller synthesis problem.

▶ **Lemma 20** (Decomposability is decidable). *Whether a game is decomposable is decidable. There exists a computable function* decomp *from games to integers such that whenever a game G is $(\preceq, k)$ decomposable for some k, it is $(\preceq, \mathrm{decomp}(G))$ decomposable.*

▶ **Theorem 21.** *The distributed synthesis problem is decidable for decomposable games.*

Proof of Theorem 21. We show that there exists a computable function f from games to integers such that in every decomposable distributed game G every strategy with no useless repetition has duration $\leq f(G)$.

Let $\preceq$ be a preorder on 2^A compatible with inclusion, k' an integer and G a $(\preceq, k')$ action decomposable distributed game. Assume $k' = \mathrm{decomp}(G)$ w.l.o.g. (cf. Lemma 20).

For every set of actions $B \subseteq A$, denote G_B the game with actions B and the same processes, initial state and final states than G. The transitions of G_B are all transitions of G whose action is in B. An action $a \in B$ is controllable in G_B iff it is controllable in G.

We show that for every $B \subseteq A$ the game G_B is $(\preceq_B, k')$ decomposable, where $\preceq_B$ denotes the restriction of $\preceq$ to 2^B. Let xy be a prime play of G_B such that y is k'-repeating. Since G is $(\preceq, k')$ decomposable, there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and a prime prefix $z \sqsubseteq y$ such that $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock in G and $C \prec \mathrm{Alph}(y)$ where $C = \{a \in A \mid \mathrm{dom}(a) \subseteq \mathbb{Q} \text{ and } a \mathbb{I} \mathrm{last}(z)\}$. Since $\preceq$ is monotonic with respect to inclusion then $\{b \in B \mid \mathrm{dom}(b) \subseteq \mathbb{Q} \text{ and } b \mathbb{I} \mathrm{last}(z)\} = (C \cap B) \preceq C \prec \mathrm{Alph}(y)$ thus $(C \cap B) \prec_B \mathrm{Alph}(y)$. Since xy is a play in G_B then $\partial_{\mathrm{last}(z)}(xz) \sqsubseteq xy$ is a play in G_B as well. And since every play in G_B is a play in G, $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock not only in G but also in G_B. All conditions of action decomposability are met : G_B is $(\preceq_B, k')$ decomposable.

Denote $R_B(m)$ the largest size of a complete undirected graph whose edges are labelled with 2^B and which contains no monochromatic clique of size $\geq m$. According to Ramsey's theorem, $R_B(m)$ is finite and computable. For every $B \subseteq A$, defined inductively $f(G_B)$ as :

$$f(G_B) = R_B\left((k' + |\mathbb{P}|) \cdot |B| \cdot |Q|^{|\mathbb{P}|} \cdot 2^{2^{|A||\mathbb{P}| \cdot \max\{f(G_{B'}), B' \prec B\}}}\right) \ ,$$

with the convention $\max \emptyset = 0$.

Fix a strategy σ with no useless repetition. We prove that for every σ-play zu,

$$|u| \leq f\left(G_{\mathrm{Alph}(u)}\right) \ . \tag{5}$$

The proof is by induction on $\mathrm{Alph}(u)$ with respect to $\preceq$. The base case when $\mathrm{Alph}(u) = \emptyset$ is easy, in this case $|u| = 0$.

Now let zu be a σ-play consistent with σ. Assume the induction hypothesis holds: for every σ-play $z'u'$, if $\mathrm{Alph}(u') \prec \mathrm{Alph}(u)$ then $|u'| \leq f\left(G_{\mathrm{Alph}(u')}\right)$.

We start with computing, for every non-empty set of letters $B \prec \mathrm{Alph}(u)$ an upper bound on the length of every factorization $u = u_0u_1 \cdots u_Nu_{N+1}$ such that

$$B = \mathrm{Alph}(u_1) = \mathrm{Alph}(u_2) = \ldots = \mathrm{Alph}(u_N) \ . \tag{6}$$

For a start, we consider the case where B is connected in the sense where the dependency graph $D_B = (B, D \cap B \times B)$ is connected. Set $k = k' + |\mathbb{P}|$. For $0 \leq \ell < \frac{N}{k}$, denote w_ℓ the concatenation $w_\ell = u_{1+\ell k} \cdot u_{2+\ell k} \cdots u_{k+\ell k}$ and $h_\ell = zu_0w_1 \ldots w_{\ell-1}$. Let $\mathbb{R}_B = \mathrm{dom}(B)$ and fix some $c \in B$.

Let $0 \leq \ell < \frac{N}{k}$. We show that $\partial_c(w_\ell)$ is k'-repeating and $\partial_c(h_\ell w_\ell) = \partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)$. Since $w_\ell = u_{1+\ell k} \cdot u_{2+\ell k} \cdots u_{k+\ell k}$, according to property (1) of views there exists a sequence $\mathbb{P} \supseteq \mathbb{R}_1 \supseteq \ldots \supseteq \mathbb{R}_k$ such that

$$\partial_c(w_\ell) = \partial_{\mathbb{R}_1}(u_{1+\ell k})\, \partial_{\mathbb{R}_2}(u_{2+\ell k}) \cdots \partial_{\mathbb{R}_k}(u_{k+\ell k}) \tag{7}$$

where $\mathbb{R}_k = \{c\}$ and for every $1 \leq i \leq k-1$, $\mathbb{R}_i = \mathbb{R}_{i+1} \cup \mathrm{dom}(\partial_{\mathbb{R}_{i+1}}(u_{i+1+\ell k}))$. Since the sequence $(\mathbb{R}_i)_{1 \leq i \leq k'+|\mathbb{P}|}$ is monotonic, there exists $i \in k' \ldots k' + |\mathbb{P}|$ such that $\mathbb{R}_i = \mathbb{R}_{i+1}$. Denote $\mathbb{R} = \mathbb{R}_i = \mathbb{R}_{i+1}$ and $B' = \{b \in B, \mathrm{dom}(b) \cap \mathbb{R} \neq \emptyset\}$ and $B'' = \{b \in B, \mathrm{dom}(b) \subseteq \mathbb{R}\}$. By definition of views, and according to (6), $B' \subseteq \mathrm{Alph}(\partial_{\mathbb{R}}(u_{i+1+\ell k}))$. Since $\mathbb{R} = \mathbb{R}_i = \mathbb{R}_{i+1}$ and $\mathbb{R}_i = \mathbb{R}_{i+1} \cup \mathrm{dom}(\partial_{\mathbb{R}_{i+1}}(u_{i+1+\ell k}))$ then $\mathrm{dom}(\partial_{\mathbb{R}}(u_{i+1+\ell k})) \subseteq \mathbb{R}$ thus $\mathrm{Alph}(\partial_{\mathbb{R}}(u_{i+1+\ell k})) \subseteq B''$. Since $B'' \subseteq B'$ then finally $B' = \mathrm{Alph}(\partial_{\mathbb{R}}(u_{i+1+\ell k})) = B''$. Thus the set B'' is a connected component of the graph $D_B = (B, D \cap B \times B)$: by definition of B' and B'', all edges with source B'' have target in $B' = B''$. However by hypothesis D_B is connected thus $B = B' = B''$ and $\mathbb{R} = \mathbb{R}_B$. Finally $\mathbb{R}_B \subseteq \mathbb{R}_i \subseteq \mathbb{R}_1$ and since $\mathbb{R}_1 \subseteq \mathrm{dom}(\partial_c(w_\ell)) \subseteq \mathbb{R}_B$, the sequence $(\mathbb{R}_i)_{1 \leq i' \leq i}$ is constant equal to $\mathbb{R}_B$. Thus, according to (6) and the definition of $\mathbb{R}_B$, for every $1 \leq i' \leq i$, $\partial_{\mathbb{R}_{i'}}(u_{i'+\ell k}) = u_{i'+\ell k}$. Thus, according to (6) and (7) and since $k' \leq i'$, every letter of B occurs at least k' times in $\partial_c(w_\ell)$ thus $\partial_c(w_\ell)$ is k'-repeating and $\partial_c(h_\ell w_\ell) = \partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)$.

Since the game is $(\preceq, k')$ decomposable and $\partial_c(w_\ell)$ is k'-repeating, and $\partial_c(h_\ell w_\ell) = \partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)$, there exists a superset $\mathbb{T}^{(\ell)}$ of $\mathbb{R}_B$, an action b_ℓ, and a prime prefix $w'_\ell b_\ell \sqsubseteq \partial_c(w_\ell)$ such that the play $z_\ell = \partial_{b_\ell}(\partial_{\mathbb{R}_B}(h_\ell) w'_\ell b_\ell)$ is a $\mathbb{T}^{(\ell)}$-lock and $B_\ell \prec B$ where $B_\ell = \{a \in A \mid \mathrm{dom}(a) \subseteq (\mathbb{T}^{(\ell)} \setminus \mathrm{dom}(b_\ell))\}$.

For every $0 \leq \ell < \frac{N}{k}$, denote $\mathrm{strate}_\ell = \left(b_\ell, (s_{\ell,p})_{p \in \mathbb{P}}, \sigma^{(\ell)}\right)$ the $\mathbb{T}^{(\ell)}$ strategic state of σ after z_ℓ. We show two properties of $(\mathrm{strate}_\ell)_{0 \leq \ell < \frac{N}{k}}$.

- First, all elements of $(\mathrm{strate}_\ell)_{0 \leq \ell < \frac{N}{k}}$ are distinct. For the sake of contradiction, assume $\mathrm{strate}_\ell = \mathrm{strate}_{\ell'}$ for some $0 \leq \ell < \ell' < \frac{N}{k}$. We show that $z_\ell \sqsubset z_{\ell'}$. Since $\mathrm{strate}_\ell = \mathrm{strate}_{\ell'}$ then $b_\ell = b_{\ell'}$, denote this letter b. Then

$$\begin{aligned} z_\ell = \partial_b(\partial_{\mathbb{R}_B}(h_\ell) w'_\ell b) \sqsubseteq \partial_b(\partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)) = \partial_b(\partial_c(h_\ell w_\ell)) \\ \sqsubseteq \partial_b(\partial_c(h_{\ell'})) \sqsubseteq \partial_b(\partial_{\mathbb{R}_B}(h_{\ell'})) \sqsubset \partial_b(\partial_{\mathbb{R}_B}(h_{\ell'}) w'_{\ell'} b) = z_{\ell'} \ , \end{aligned}$$

  where the second inequality holds because $h_\ell w_\ell \sqsubseteq h_{\ell'}$ since $\ell \leq \ell' - 1$, and the third inequality holds because $c \in B$ thus $\mathrm{dom}(c) \subseteq \mathbb{R}_B$ hence property (2) applies. Moreover the last inequality is strict because there is at least one more b in $\partial_b(\partial_{\mathbb{R}_B}(h_{\ell'}) w'_{\ell'} b)$ than in $\partial_b(\partial_{\mathbb{R}_B}(h_{\ell'}))$. We get a contradiction because by hypothesis there is no useless repetition in σ, however, denoting $x = z_\ell$ and y such that $xy = z_{\ell'}$, the pair (x, y) is a useless $\mathbb{T}^{(\ell)}$-repetition in σ: by hypothesis the strategic $\mathbb{T}^{(\ell)}$-states of z_ℓ and $z_{\ell'}$ are equal and both x and xy are $\mathbb{T}^{(\ell)}$-locks, moreover y is not empty because $z_\ell \sqsubset z_{\ell'}$ and finally $\mathrm{dom}(y) \subseteq \mathrm{dom}(u_{1+\ell k} \cdots u_{k+\ell' k}) \subseteq \mathbb{R}_B \subseteq \mathbb{T}^{(\ell)}$. Thus (x, y) is a useless repetition in σ.

- Second, for every $0 \leq \ell < \frac{N}{k}$, all plays in $\sigma^{(\ell)} = \pi(\sigma, z_\ell, \mathbb{T}^{(\ell)} \setminus \mathrm{dom}(b_\ell))$ have length $\leq m = \max_{B' \prec B} f(G_{B'})$. Let $z_\ell u'$ be a σ-play such that $\mathrm{dom}(u') \subseteq (\mathbb{T}^{(\ell)} \setminus \mathrm{dom}(b_\ell))$. Then $\mathrm{Alph}(u') \subseteq B_\ell$. Since $\preceq$ is monotonic with respect to inclusion, $\mathrm{Alph}(u') \preceq B_\ell \prec B \preceq \mathrm{Alph}(u)$. Thus by induction hypothesis, $|u'| \leq f\left(G_{\mathrm{Alph}(u')}\right) \leq m$.

According to the second property, there are at most $2^{2^{m|A||\mathbb{P}|}}$ different residuals appearing in the sequence $(\sigma^{(\ell)})_{0 \leq \ell < \frac{N}{k}}$. Thus the sequence $(\mathrm{strate}_\ell)_{0 \leq \ell < \frac{N}{k}}$ takes at most $K = |B| \cdot |Q|^{|\mathbb{P}|} \cdot 2^{2^{m|A||\mathbb{P}|}}$ different values. And according to the first property, all these states are different thus $N \leq k \cdot K$.

The inequality $N \leq k \cdot K$ has been established under the assumption that D_B is connected. The general case reduces to this case: let C be a connected component of D_B and for $1 \leq i \leq N$ let v_i be the projection of u_i on C. Then $\forall 1 \leq i \leq N, \mathrm{Alph}(v_i) = C$ and there exists u'_0 such that $u = u'_0 v_1 v_2 \dots v_N u_{N+1}$ thus $N \leq k \cdot K$.

Let us reformulate the inequality $N \leq k \cdot K$ as a property of an undirected complete graph with edges colored by 2^A. Let $u = a_1 a_2 \cdots a_{|u|}$ the factorization of u into its letters. Let J_u be the complete graph with vertices $1, \dots, |u|$ and the label of the edge $\{i, j\}$ with $i < j$ is the set of letters $\{a_i, \dots, a_j\}$. Then every monochromatic clique of J_u has size $\leq k \cdot K$. Thus, according to Ramsey theorem, $|u| \leq R_{\mathbb{T}}(k \cdot K) = R_{\mathbb{T}}((k' + |\mathbb{P}|) \cdot K)$, which completes the inductive step.

As a consequence, winning strategies in G can be looked for in the finite family of strategies all of whose plays have length $\leq f(G)$ with $f(G)$ computable. As a consequence, the synthesis problem can be solved by enumerating all these strategies and testing whether any of them is winning. For testing whether a strategy of finite duration is winning the algorithm simply checks that the global state of all the maximal plays is final. ◀

5.3 New examples of decidable games

The three classes of games whose decidability is already known are decomposable (cf Lemmas 14, 16 and 18). In this section we give some new examples of decidable games.

▶ **Lemma 22.** *Four players games are structurally decomposable.*

Although our techniques do not seem to provide an algorithm for solving games with five processes, they can address a subclass.

▶ **Lemma 23.** *Let G be a distributed game with five processes. Assume that the number of actions that a process can successively play in a row without synchronizing simultaneously with two other processes is bounded. Then G is process decomposable.*

Another decidable example is the class of *majority games*:

▶ **Lemma 24** (Majority games)**.** *Assume that every non-local action synchronizes a majority of the processes i.e. for every action a, $|\mathrm{dom}(a)| = 1$ or $|\mathrm{dom}(a)| \geq |\mathbb{P} \setminus \mathrm{dom}(a)|$. Then the game is structurally decomposable.*

The class of structurally decomposable games is stable under projection and merge.

▶ **Definition 25** (Projecting games)**.** Let G be a game with processes $\mathbb{P}$ and alphabet $(A_p)_{p \in \mathbb{P}}$. Let $\mathbb{P}' \subseteq \mathbb{P}$ a subset of the processes. The projection of G on $\mathbb{P}'$ is the game G' with processes $\mathbb{P}'$ and alphabet $A' = \{a \in A \mid \mathrm{dom}(a) \cap \mathbb{P}' \neq \emptyset\}$ partitioned in $(A' \cap A_p)_{p \in \mathbb{P}'}$. The states of a process $p \in \mathbb{P}'$ are the same in G and G', every transition $\delta \in \{a\} \times \Pi_{p \in \mathrm{dom}(a)} Q_p \times Q_p$ of G on a letter $a \in A'$ is projected to $\{a\} \times \Pi_{p \in \mathrm{dom}(a) \cap \mathbb{P}'} Q_p \times Q_p$, and every transition on a letter $a \notin A'$ is simply deleted.

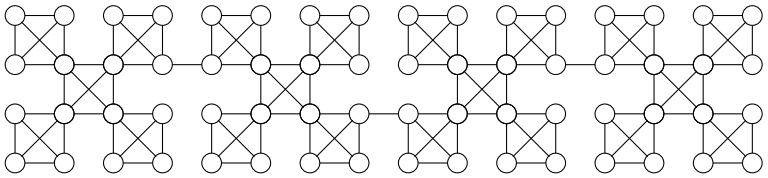


Figure 2 A decidable process architecture.

The following result combines two structurally decomposable games into one.

▶ **Lemma 26** (Merging games). *Let G be a game, and $\mathbb{P}_0, \mathbb{P}_1 \subseteq \mathbb{P}$ two set of processes such that $\mathbb{P} = \mathbb{P}_0 \cup \mathbb{P}_1$ and $\mathbb{P}_0 \cap \mathbb{P}_1 \neq \emptyset$ and for every action $a \in A$,*

$$(\mathrm{dom}(a) \cap \mathbb{P}_0 \neq \emptyset) \wedge (\mathrm{dom}(a) \cap \mathbb{P}_1 \neq \emptyset) \implies (\mathbb{P}_0 \cap \mathbb{P}_1 \subseteq \mathrm{dom}(a)) \ .$$

If both projections of G on $(\mathbb{P}_0 \setminus \mathbb{P}_1)$ and $(\mathbb{P}_1 \setminus \mathbb{P}_0)$ are structurally decomposable then G is structurally decomposable.

The merge operation can combine two structurally decomposable games in order to create a new one. For example all acyclic games can be obtained this way, since 3-player games are structurally decomposable and every tree with more than three nodes can be obtained by merging two strictly smaller subtrees. This technique can go beyond acyclic games, by merging together 4-player games and majority games. The graph of processes is an undirected graph with nodes $\mathbb{P}$ and there is an edge between p and q whenever both p and q both belong to the domain of one of the actions. Then all the games whose graph of processes is contained in the one depicted on Fig. 2 are structurally decomposable.

6 Conclusion

We considered the DISTRIBUTED SYNTHESIS PROBLEM, which aims at controlling asynchronous automata using automatically synthesized controllers with causal memory. We presented a theorem that unifies several known decidability results and provide new ones.

The decidability of this problem is, to the best of our knowledge, stil opened, even in the simple case where the graph of processes is a ring of five processes where each process can interact only with both its neighbors.

Another intriguing open problem is the case of *weakly k-connectedly communicating* plants. In such a plant, whenever two processes play both k times in a row without hearing from each other, they will never hear from each other anymore. It is not known whether the MSO theory of the corresponding event structures is decidable or not [6], neither do we know how to use techniques of this paper to solve this class of games.

Acknowledgements. We thank Blaise Genest, Anca Muscholl, Igor Walukiewicz, Paul Gastin and Marc Zeitoun for interesting discussions on the topic. Moreover we thank one of the reviewers of a previous version, who spotted several mistakes and did provide very useful comments which led to several improvements in the presentation of the results. We also thank Engel Lefaucheux for spotting a missing hypothesis in Lemma 26.

References

1 Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995. URL: `https://books.google.co.uk/books?id=vNFLOE2pjuAC`.

2 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 321–330. IEEE, 2005.

3 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

4 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

5 Hugo Gimbert. A class of zielonka automata with a decidable controller synthesis problem. *CoRR*, abs/1601.05176, 2016. arXiv:1601.05176.

6 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005. doi:10.1007/11590156_16.

7 Anca Muscholl. Automated synthesis of distributed controllers. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *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 11–27. Springer, 2015. doi:10.1007/978-3-662-47666-6_2.

8 Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 639–651. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

9 Anca Muscholl, Igor Walukiewicz, and Marc Zeitoun. A look at the control of asynchronous automata. In M. Mukund K. Lodaya and eds. N. Kumar, editors, *Perspectives in Concurrency Theory*. Universities Press, CRC Press, 2009.

10 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 746–757. IEEE, 1990.

11 Peter JG Ramadge and W Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

12 Wieslaw Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.

Streaming for Aibohphobes: Longest Palindrome with Mismatches

Elena Grigorescu[1], Erfan Sadeqi Azer[2], and Samson Zhou[3]

1 **Department of Computer Science, Purdue University, West Lafayette, USA**
elena-g@purdue.edu.
2 **School of Informatics and Computing, Indiana University, Bloomington, USA**
esadeqia@indiana.edu.
3 **Department of Computer Science, Purdue University, West Lafayette, USA**
samsonzhou@gmail.com.

Abstract

A palindrome is a string that reads the same as its reverse, such as "aibohphobia" (fear of palindromes). Given a metric and an integer $d > 0$, a *d-near-palindrome* is a string of Hamming distance at most d from its reverse.

We study the natural problem of identifying the longest d-near-palindrome in data streams. The problem is relevant to the analysis of DNA databases, and to the task of repairing recursive structures in documents such as XML and JSON.

We present the first streaming algorithm for the longest d-near-palindrome problem that returns a d-near-palindrome whose length is within a multiplicative $(1+\epsilon)$-factor of the longest d-near-palindrome. Our algorithm also returns the set of mismatched indices in the d-near-palindrome, and uses $\mathcal{O}\left(\frac{d\log^7 n}{\epsilon\log(1+\epsilon)}\right)$ bits of space, and $\mathcal{O}\left(\frac{d\log^6 n}{\epsilon\log(1+\epsilon)}\right)$ update time per arrival symbol. We show that for $d = o(\sqrt{n})$, any randomized algorithm with multiplicative approximation $(1+\epsilon)$ that succeeds with probability at least $1 - 1/n$ requires $\Omega(d\log n)$ space.

We further obtain a streaming algorithm that returns a d-near-palindrome whose length is within an additive E-error of the longest d-near-palindrome. The algorithm uses $\mathcal{O}\left(\frac{dn\log^6 n}{E}\right)$ bits of space and $\mathcal{O}\left(\frac{dn\log^5 n}{E}\right)$ update time. As before, we show that any randomized streaming algorithm that solves the longest d-near-palindrome problem for additive error E with probability at least $1 - \frac{1}{n}$, uses $\Omega\left(\frac{dn}{E}\right)$ space.

Finally, we give an *exact* two-pass algorithm that solves the longest d-near-palindrome problem using $\mathcal{O}\left(d^2\sqrt{n}\log^6 n\right)$ bits of space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Longest palindrome with mismatches, Streaming algorithms, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.31

1 Introduction

A palindrome is a string that reads the same as its reverse, such as the common construct "racecar", or the deliberate construct "aibohphobia". Given a metric and an integer $d > 0$, we say that a string is a *d-near-palindrome* if it is at distance at most d from its reverse. In this paper, we study the problem of identifying the longest d-near-palindrome substring in the *streaming* model, under the Hamming distance. In the streaming model, the input data is streamed one symbol at a time, and we are allowed to perform computation using

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 31; pp. 31:1–31:13

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

only a small amount of working memory. Specifically, our goal is to approximate the length of the longest near-palindrome in a string of length n, using only $o(n)$ space. A related question regarding approximating the length of the longest palindrome in RNA sequences under removal of elements was explicitly asked at the Bertinoro Workshop on Sublinear Algorithms 2014 [1].

Finding near-palindromes is widely motivated in string processing of databases relevant to bioinformatics.

Specifically, since the development of the Human Genome Project, advances in biological algorithms have quickened the sequencing for genes and proteins, leading to increasingly large databases of strings representing both nucleic acids for DNA or RNA, and amino acids for proteins. Tools to analyze these sequences, such as the basic local alignment search tool (BLAST) [2] often require the removal of "low-complexity" regions (long repetitive or palindromic structures). However, these long sequences frequently contain small perturbations through mutation or some other form of corruption (including human error), so that identifying "near"-palindromes under either Hamming distance or edit distance is important for preprocessing sequences before applying the heuristic tools. In particular, the streaming model is relevant to contemporary data-sequencing technologies for near-palindromes, as further discussed in [9, 12].

Our contributions

We initiate the study of finding near-palindromes in the streaming model, and provide several algorithms for the longest near-palindrome substring.

Given a stream S of length n and an integer $d = o(\sqrt{n})$, let ℓ_{max} be the length of a longest d-near-palindrome substring in S.

▶ **Theorem 1.** *There exists a one-pass streaming algorithm that returns a d-near-palindrome of length at least $\frac{1}{1+\epsilon} \cdot \ell_{max}$, with probability $1 - \frac{1}{n}$. The algorithm uses $\mathcal{O}\left(\frac{d \log^7 n}{\epsilon \log(1+\epsilon)}\right)$ bits of space and update time $\mathcal{O}\left(\frac{d \log^6 n}{\epsilon \log(1+\epsilon)}\right)$ per arriving symbol.*

▶ **Theorem 2.** *There exists a one-pass streaming algorithm that returns a d-near-palindrome of length at least $\ell_{max} - E$, with probability $1 - \frac{1}{n}$. The algorithm uses $\mathcal{O}\left(\frac{dn \log^6 n}{E}\right)$ bits of space and update time $\mathcal{O}\left(\frac{dn \log^5 n}{E}\right)$ per arriving symbol.*

If two passes over the stream are allowed, one can find an *exact* longest d-near-palindrome.

▶ **Theorem 3.** *There exists a two-pass streaming algorithm that returns a d-near-palindrome of length ℓ_{max}, with probability $1 - \frac{1}{n}$. It uses $\mathcal{O}\left(d^2 \sqrt{n} \log^6 n\right)$ bits of space and $\mathcal{O}\left(d^2 \sqrt{n} \log^5 n\right)$ update time per arriving symbol.*

We complement our results with lower bounds for randomized algorithms.

▶ **Theorem 4.** *Let $d = o(\sqrt{n})$. Then any randomized streaming algorithm that returns a d-near-palindrome of length at least $\frac{\ell_{max}}{1+\epsilon}$ with probability at least $1 - \frac{1}{n}$ must use $\Omega\left(d \log n\right)$ bits of space.*

▶ **Theorem 5.** *Let $d = o(\sqrt{n})$ and $E > d$ be an integer. Then any randomized streaming algorithm that returns a d-near-palindrome of length at least $\ell - E$, with probability at least $1 - \frac{1}{n}$ must use $\Omega\left(\frac{dn}{E}\right)$ bits of space.*

A summary of our results and comparison with related work appears in Table 1.

Table 1 Summary of our results and comparison to related work

	Space for Algorithms		Lower Bounds	
Model	d-Near-Palindrome	Palindrome	d-Near-Palindrome	Palindrome
1-Pass, Multiplicative $(1+\epsilon)$	$\mathcal{O}\left(\frac{d\log^7 n}{\epsilon\log(1+\epsilon)}\right)$	$\mathcal{O}\left(\frac{\log^2 n}{\epsilon\log(1+\epsilon)}\right)$ [5]	$\Omega(d\log n)$	$\Omega\left(\frac{\log n}{\log(1+\epsilon)}\right)$ [10]
1-Pass, Additive E	$\mathcal{O}\left(\frac{dn\log^6 n}{E}\right)$	$\mathcal{O}\left(\frac{n\log n}{E}\right)$ [5]	$\Omega\left(\frac{dn}{E}\right)$	$\Omega\left(\frac{n}{E}\right)$ [5]
2-Pass, Exact	$\mathcal{O}\left(d^2\sqrt{n}\log^6 n\right)$	$\mathcal{O}\left(\sqrt{n}\log n\right)$ [5]	-	-

Background and Related Work

Our techniques extend previous work on the Longest Palindromic Substring Problem, the Pattern Matching Problem, and the d-Mismatch Problem in the streaming model.

In the *Longest Palindromic Substring Problem*, the goal is to output a longest palindromic substring of an input of length n, while minimizing the computation space. Manacher [14] introduces a linear-time online algorithm, which reports whether all symbols seen at the time of query form a palindrome. Berenbrink *et al.* [5] achieve $\mathcal{O}\left(\frac{\log^2 n}{\epsilon\log(1+\epsilon)}\right)$ space for multiplicative error $(1+\epsilon)$, and show a space lower bound for algorithms with additive error. Gawrychowski *et al.* [10] recently generalize the aforementioned lower bounds for additive error, and also produce a space lower bound of $\Omega\left(\frac{\log n}{\log(1+\epsilon)}\right)$ for algorithms with multiplicative error $(1+\epsilon)$, which is essentially tight.

In the *Pattern Matching Problem*, one is given a pattern of length m and the goal is to output all occurrences of the pattern in the input string, while again minimizing space or update time. In order to achieve space sublinear in the size of the input, many pattern matching streaming algorithms use Karp-Rabin fingerprints [13]. Porat and Porat [15] present a randomized algorithm for exact pattern matching using $\mathcal{O}(\log m)$ space and $\mathcal{O}(\log m)$ update time, which Breslauer and Galil [6] further improve to constant update time.

In the *d-Mismatch Problem*, one is given a pattern and the goal is to find all substrings of the input that are at most Hamming distance d from the pattern. A line of exciting work (e.g., [3, 15, 7, 4]) culminates in a recent algorithm by Clifford *et al.* [8] that uses $\mathcal{O}\left(d^2\mathsf{polylog}\, m\right)$ space and $\mathcal{O}\left(\sqrt{d}\log d + \mathsf{polylog}\, m\right)$ update time per arriving symbol.

2 Preliminaries

We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. We assume an input stream of length n over alphabet Σ. Given a string $S[1, n]$, we denote its length by $|S|$, its i^{th} character by $S[i]$, and the substring between locations i and j (inclusive) by $S[i, j]$.

The Hamming distance between S and T, denoted $\mathsf{HAM}(S, T)$ is the number of indices whose symbols do not match: $\mathsf{HAM}(S, T) = \left|\{i \mid S[i] \neq T[i]\}\right|$. We denote the concatenation of S and T by $S \circ T$. Each index i such that $S[i] \neq S[n - i + 1]$ is a *mismatch*. We say S is a *d-near-palindrome* if $\mathsf{HAM}(S, S^R) \leq d$. Without loss of generality, our algorithms assume the lengths of d-near-palindromes are even, since for any odd length d-near-palindrome, we may apply the algorithm to $S[1]S[1]S[2]S[2]\cdots S[n]S[n]$ instead of $S[1, n]$.

▶ **Definition 6** (Karp-Rabin Fingerprint)**.** For a string S, a prime P and an integer B with $1 \leq B < P$, the Karp-Rabin forward and reverse fingerprints [13] are defined as follows:

$$\phi^F(S) = \left(\sum_{x=1}^{|S|} S[x]\cdot B^x\right) \bmod P, \qquad \phi^R(S) = \left(\sum_{x=1}^{|S|} S[x]\cdot B^{-x}\right) \bmod P.$$

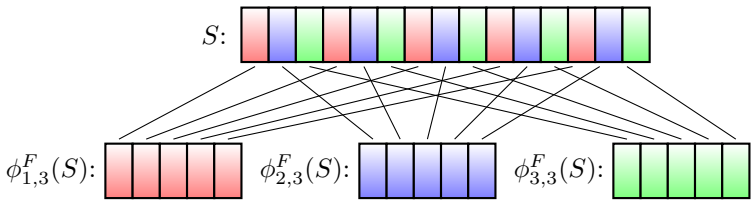


Figure 1 Karp-Rabin Fingerprints for first-level subpattern.

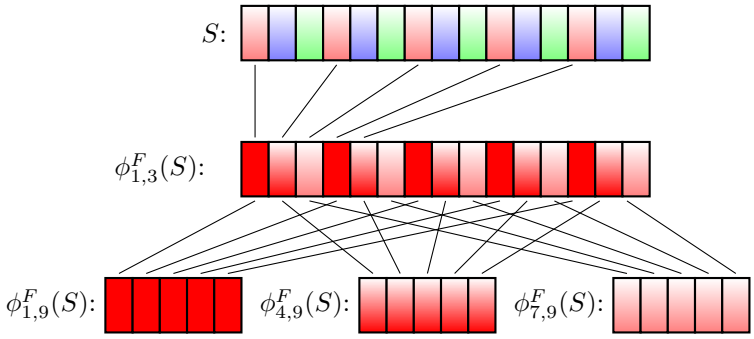


Figure 2 Karp-Rabin Fingerprints for second-level subpattern.

Karp-Rabin Fingerprints have the following easily verifiable properties:

1. $\phi^R(S) \cdot B^{|S|+1} = \phi(S^R) \bmod P$
2. $\phi^F(S[x,y]) = B^{1-x}(\phi^F(S[1,y]) - \phi^F(S[1,x-1])) \bmod P$
3. $\phi^R(S[x,y]) = B^{x-1}(\phi^R(S[1,y]) - \phi^R(S[1,x-1])) \bmod P$

We use Karp-Rabin Fingerprints for certain subpatterns of S, as in [8]. For a string S and integers $a \le b$, define the *first-level subpattern* $S_{a,b}$ to be the subsequence $S[a]S[a+b]S[a+2b]\ldots$. In this case, define $S^R_{a,b} = (S_{a,b})^R$ (as opposed to $(S^R)_{a,b}$). Similarly, define $S_{a,b}[x,y] = S_{a,b} \cap S[x,y]$ (as opposed to $(S[x,y])_{a,b}$). Then for $1 \le a \le b$, define the fingerprints for $S_{a,b}$ and its reverse:

$$\phi^F_{a,b}(S) = \phi^F(S_{a,b}) = \left(\sum_{x \equiv a \bmod b} S[x] \cdot B^{\lceil x/b \rceil} \right) \bmod P$$

$$\phi^R_{a,b}(S) = \phi^R(S_{a,b}) = \left(\sum_{x \equiv a \bmod b} S[x] \cdot B^{-\lceil x/b \rceil} \right) \bmod P$$

For an example, see Figure 1.

Given a first-level subpattern $T = S_{a,b} = S[a]S[a+b]S[a+2b]\ldots$ and integers $r \le s$, define the *second-level subpattern* $T_{r,s} = T[r]T[r+s]T[r+2s]\ldots$. Observe that $T_{r,s} = S_{a+rb,sb}$ and thus, second-level subpatterns are simply more refined first-level subpatterns. For an example, see Figure 2.

Observe the following properties of fingerprints on first-level and second-level subpatterns:

1. $\phi^R_{a,b}(S) \cdot B^{|S|+1} = \phi_{|S|-a+1,b}(S^R) \bmod P$
2. $\phi^F_{a,b}(S[x,y]) = B^{\lceil (1-x)/b \rceil}(\phi^F_{a,b}(S[1,y]) - \phi^F_{a,b}(S[1,x-1])) \bmod P$
3. $\phi^R_{a,b}(S[x,y]) = B^{\lceil (x-1)/b \rceil}(\phi^R_{a,b}(S[1,y]) - \phi^R_{a,b}(S[1,x-1])) \bmod P$

We also use the following application of the Prime Number Theorem:

▶ **Lemma 7.** *(Adaptation of Lemma 4.1 [8]) Given two distinct integers $a, b \in [1,n]$ and a random prime number $p \in \left[\frac{d}{\beta}\log^2 n, \frac{34d}{\beta}\log^2 n\right]$ where $\beta = \frac{1}{16}$, then $\mathbf{Pr}\left[a \equiv b \bmod p\right] \le \frac{\beta}{32d}$.*

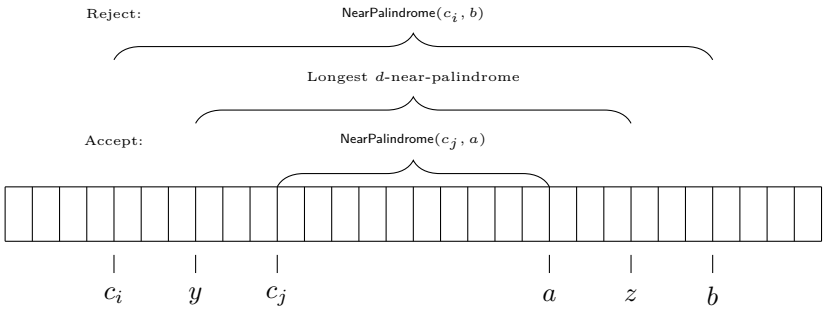


Figure 3 The longest d-near-palindrome will be sandwiched within checkpoints to provide a $(1+\epsilon)$-approximation of ℓ_{max}. That is, $(1+\epsilon)(a-c_j) \geq (z-y)$.

Finally, we remark that problems in bioinformatics, such as the *RNA Folding Problem* [1], use the following notion of complementary palindromes:

▶ **Definition 8.** Let $f : \sum \to \sum$ be a pairing of symbols in the alphabet. A string $S \in \sum^n$ is a complementary palindrome if $S[x] = f(S[n+1-x])$ for all $1 \leq x \leq n$.

Our algorithms can be modified to recognize complementary palindromes with the same space usage and update time. Indeed, we only need to modify the forward fingerprints to use $f(S[x])$ instead of $S[x]$:

$$\phi^F_{a,b}(S) = \left(\sum_{x \equiv a \bmod b} f(S[x]) \cdot B^{\lceil x/b \rceil} \right) \bmod P.$$

3 Overview and Techniques

One-pass Multiplicative Approximation Algorithm

Our algorithm combines and extends ideas and techniques from the solution to the d-Mismatch Problem in [8] and the solution to the Longest Palindrome Problem in [5]. For additional clarity, we provide several figures in [11].

As the stream progresses, we keep a set of checkpoints $\mathcal{C}$, where $c \in \mathcal{C}$ is the beginning of potential d-near-palindromes that we output. We also maintain a sliding window that contains the $2d$ most recently seen symbols. The sliding window identifies any d-near-palindrome of length at most $2d$. It also guesses that the midpoint of the sliding window is the midpoint of a potential d-near-palindrome of length $> 2d$.

We keep an estimate $\tilde{\ell}$ of the length ℓ_{max} of the longest d-near-palindrome seen throughout the stream, as well as the starting index c_{start} of the d-near-palindrome, and the locations of the mismatches, a set of size at most d. Upon reading symbol $S[x]$ of the stream, we call procedure NearPalindrome to see if $S[c_i, x]$ is a d-near-palindrome, for each checkpoint c_i such that $x - c_i > \tilde{\ell}$. Using the framework of [5], we create and update checkpoints throughout the stream so that we find a d-near-palindrome of length at least $\frac{\ell_{max}}{1+\epsilon}$, as in Figure 3.

The algorithm in [5] maintains a list of potential midpoints associated with each checkpoint. This list can be linear in size, however it satisfies nice structural results that can be used to succinctly represent the list of candidate midpoints. Directly adapting these structural results to our setting would incur an extra factor of d in our space complexity. We avoid this extra factor by circumventing the list of candidate midpoints in the one-pass algorithms altogether.

We now overview the procedure NearPalindrome that we use repeatedly in our algorithms. It returns whether $S[c_i, x]$ is a d-near-palindrome, and if so it returns the mismatches.

The procedure NearPalindrome follows along the lines of the d-mismatch streaming algorithm from [8]. Recall that in the d-Mismatch Problem, we are given a pattern R and a text S and the algorithm is required to output a string $S[x - |R| + 1, x]$ such that $\mathsf{HAM}(R, S[x - |R| + 1, x]) \leq d$. While in the d-Mismatch Problem the pattern is fixed, here we essentially use a variable-length pattern. Namely, we check if for checkpoint c_i it is the case that $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) \leq d$. To achieve this, we maintain a dynamic set of fingerprints that we compare against the dynamic set of text fingerprints.

The procedure has two stages. In the first stage it eliminates strings T with $\mathsf{HAM}(T, T^R) \geq 2d$, while in the second stage it eliminates strings with $d < \mathsf{HAM}(T, T^R) < 2d$. This can be achieved by estimating the distance between T and T^R using fingerprints of equivalence classes modulo different primes.

Picking random primes should distribute the mismatches into different equivalence classes. The procedure estimates the number of mismatches by comparing the fingerprints of the substrings whose indices are in the same congruence class modulo p with the reverse fingerprints, namely $T_{r,p}$ and $T^R_{r,p}$ for all $1 \leq r \leq p$. Denote by $T_{r,p}$ and $T^R_{r,p}$ the *first-level fingerprints*.

By the second stage we are only left with the strings with a small number of mismatches. In order to recover the mismatches, one needs to refine each subpattern $\tilde{T} = T_{r,p}$ by picking smaller primes p', and comparing the fingerprints of the strings $\tilde{T}_{r',p'}$ and $\tilde{T}^R_{r',p'}$ for all $1 \leq r' \leq p'$. Denote by $\tilde{T}_{r',p'}$ and $\tilde{T}^R_{r',p'}$ the *second-level fingerprints* (see Figure 2).

In the first stage, we sample $2 \log n$ primes uniformly at random from $\left[\frac{d}{\beta} \log^2 n, \frac{34d}{\beta} \log^2 n\right]$, where $\beta = 1/16$. Each prime generates p subpatterns containing positions in the same equivalence class (mod p). Therefore, there are $\mathcal{O}\left(d \log^3 n\right)$ first-level subpatterns. In the second stage, we take all primes in $[\log n, 3 \log n]$ that together with the primes picked in the first stage generate a total of $\mathcal{O}\left(d \log^5 n\right)$ second-level subpatterns.

Finally, we assume throughout the paper that the fingerprints of any subpattern do not fail. Since there are at most n^3 subpatterns, and the probability that a particular fingerprint fails is at most $\frac{1}{n^5}$ for $P \in [n^5, n^6]$ (by Theorem 1 in [6]), then by a union bound, the probability that no fingerprint fails is least $1 - \frac{1}{n^2}$.

Our choice of parameters is more space-efficient compared to the data structure given by [8], which uses $\mathcal{O}\left(d^2 \log^7 n\right)$ space, since we no longer need the sliding functionality provided by their data structure. Also, the data structure given by [16] does not suffice, as it does not support concatenation, which is needed for maintaining the checkpoints.

One-pass Additive Approximation and Two-Pass Exact Algorithms

To obtain the one-pass additive approximation, we modify our checkpoints, so that they appear in every $\left\lfloor \frac{E}{2} \right\rfloor$ positions. Hence, the longest d-near-palindrome must have some checkpoint within $\left\lfloor \frac{E}{2} \right\rfloor$ positions of it, and the algorithm will recover a d-near-palindrome with length at least $\ell_{max} - E$.

To obtain the two-pass *exact* algorithm, we set $E = \sqrt{n}$ and modify the additive error algorithm so that it returns a list $\mathcal{L}$ of candidate midpoints of d-near-palindromes. Moreover, we show a structural result in Lemma 16, which allows us to compress certain substrings in the first pass, so that the second pass can recover mismatches for any potential d-near-palindromes within these substrings.

In the second pass, we carefully keep track of the $\frac{\sqrt{n}}{2}$ characters before the starting positions of long d-near-palindromes identified in the first pass. We use the compressed

information from the first pass to reconstruct the fingerprints and calculate the number of mismatches within these long d-near-palindromes identified in the first pass. However, the actual d-near-palindromes may extend beyond the estimate returned in the first pass. Thus, we compare the $\frac{\sqrt{n}}{2}$ characters after the d-near-palindromes identified in the first pass with the $\frac{\sqrt{n}}{2}$ characters that we track. This allows us to exactly identify the longest d-near-palindrome during the second pass.

Lower Bounds

To show lower bounds for randomized algorithms solving the d-near palindrom problem we use Yao's Principle [17], and construct distributions for which any deterministic algorithm fails with significant probability unless given a certain amount of space. We first reduce the problem of approximating longest d-near-palindromes to the problem of exactly identifying whether two strings have Hamming distance at most d. We then carefully construct hard distributions, and show via counting arguments that deterministic algorithms using a little of space will fail with significant probability on inputs from these distributions. We also use some ideas from [10] who showed lower bounds for streaming palindromes. Due to space constraints, the proofs of Theorem 4 and Theorem 5 are detailed in [11].

4 One-Pass Streaming Algorithm with Multiplicative Error $(1+\epsilon)$

In this section, we prove Theorem 1. Namely, we provide a one-pass streaming algorithm with multiplicative error $(1+\epsilon)$, using space $\mathcal{O}\left(\frac{d\log^7 n}{\epsilon\log(1+\epsilon)}\right)$ bits of space.

4.1 Algorithm

As described in the overview, similar to [5], we maintain a sliding window of size $2d$, along with master fingerprints, and a series of checkpoints. From the sliding window, we observe any d-near-palindrome with length at most $2d$, as well as any candidate midpoints. Then prior to seeing element $S[x]$ in the stream, we keep the following in memory:

Initialization:

1. Pick a prime P from $[n^5, n^6]$ and an integer $B < P$ (the modulo and the base of the Karp-Rabin fingerprints, respectively).
2. For the first-level fingerprints, create a set $\mathcal{P}$ consisting of $2\log n$ primes $p_1, p_2, \ldots, p_{2\log n}$ sampled independently and uniformly at random from $\left[\frac{d}{\beta}\log^2 n, \frac{34d}{\beta}\log^2 n\right]$, where $\beta = \frac{1}{16}$.
3. For the second-level fingerprints, let $\mathcal{Q}$ be the set of all primes in $[\log n, 3\log n]$.
4. Initialize a sliding window of size $2d$.
5. Initialize the sets of *Master Fingerprints*, $\mathcal{F}^F$ and $\mathcal{F}^R$:
   a. Set $\phi^F_{r,p}(S) = 0$, $\phi^R_{r,p}(S) = 0$ for all $p \in \mathcal{P}$ and $1 \le r \le p$.
   b. Set $\phi^F_{r',pq}(S) = 0$, $\phi^R_{r',pq}(S) = 0$ for all $p \in \mathcal{P}$, $q \in \mathcal{Q}$ and $1 \le r' \le pq$.
   c. Let $\mathcal{F}^F$ be the set of all $\phi^F(S)$.
   d. Let $\mathcal{F}^R$ be the set of all $\phi^R(S)$.
6. Set $k_0 = \frac{\log(1/\alpha)}{\log(1+\alpha)}$, where $\alpha = \sqrt{1+\epsilon} - 1$.
7. Initialize a list of checkpoints $\mathcal{C} = \emptyset$.
8. Set the starting index c_{start} to be 1, the length estimate $\tilde{\ell}$ of the longest d-near-palindrome found so far to be 0, and the at most d mismatched indices $\mathcal{M} = \emptyset$.

We now formalize the steps outlined in the overview. The data structure relies on the procedure NearPalindrome that we describe and analyze in detail in Section 4.2.

Maintenance:

1. Read $S[x]$. Update the sliding window to $S[x-2d, x]$.
2. Update the *Master Fingerprints* to be $\mathcal{F}^F(1,x)$ and $\mathcal{F}^R(1,x)$:
   a. Update the first-level fingerprints: for every $p \in \mathcal{P}$, let $r \equiv x \mod p$, and increment $\phi^F_{r,p}(S)$ by $S[x] \cdot B^{\lceil x/p \rceil} \mod P$ and increment $\phi^R_{r,p}(S)$ by $S[x] \cdot B^{-\lceil x/p \rceil} \mod P$.
   b. Update the second-level fingerprints: for every $p \in \mathcal{P}$ and $q \in \mathcal{Q}$, let $r' \equiv x \mod pq$, and increment $\phi^F_{r',pq}(S)$ by $S[x] \cdot B^{\lceil x/(pq) \rceil} \mod P$ and increment $\phi^R_{r',pq}(S)$ by $S[x] \cdot B^{-\lceil x/(pq) \rceil} \mod P$.
3. For all $k \geq k_0$:
   a. If x is a multiple of $\lfloor \alpha(1+\alpha)^{k-2} \rfloor$, then add the checkpoint $c = x$ to $\mathcal{C}$. Set $level\,(c) = k$, $fingerprints\,(c) = \mathcal{F}^F(1,x) \cup \mathcal{F}^R(1,x)$.
   b. If there exists a checkpoint c with $level\,(c) = k$ and $c < x - 2(1+\alpha)^k$, then delete c from $\mathcal{C}$.
4. For every checkpoint $c \in \mathcal{C}$ such that $x - c > \tilde{\ell}$, we call NearPalindrome (described in Section 4.2) to see if $S[c,x]$ is a d-near-palindrome. If $S[c,x]$ is a d-near-palindrome, then set $c_{start} = c$, $\tilde{\ell} = x - c$ and $\mathcal{M}$ to be the indices returned by NearPalindrome.
5. If $x = n$, then report c_{start}, $\tilde{\ell}$, and $\mathcal{M}$.

4.2 NearPalindrome and its analysis

In this section, we describe and analyze the randomized procedure NearPalindrome that receives as input a string, and decides whether it is a d-near-palindrome or not. Moreover, if the string is a d-near-palindrome, NearPalindrome returns the locations of the mismatched indices. As mentined, NearPalindrome adapts ideas from [8] for solving the d-mismatch problem. Our proofs of the properties of NearPalindrome follow almost verbatim from the statements in [8], with the only difference being that we make the magnitudes of the chosen primes as large as to withstand patterns of length $\mathcal{O}(n)$. For completeness, we include all the proofs in [11].

Before formally describing NearPalindrome we introduce some notation. Given a string $S[x,y]$, and prime p_j let $\Delta_j(x,y)$ be the number of $r \in [p_j]$ such that the subpatterns $S_{r,p_j}[x,y]$ and $S^R_{r,p_j}[x,y]$ are different. Note that we can compute $\Delta_j(x,y)$ from the fingerprints $\mathcal{F}^F(x,y)$ and $\mathcal{F}^R(x,y)$ as the number of indices r such that $\phi^F_{r,p_j}[x,y] \neq B^{k+1} \cdot \phi^R_{r,p_j}[x,y] \mod P$, where k is the length of $S_{r,p_j}[x,y]$. Define $\Delta(x,y) = \max_j \Delta_j(x,y)$. We may assume throughout that $S[x,y]$ has even length. Next we summarize some useful properties of $\Delta(x,y)$.

▶ **Lemma 9.** *(Adaptation of Lemma 5.1 and Lemma 5.2 [8]) Let $\beta = 1/16$.*

1. *If $\mathsf{HAM}(S[x,y], S^R[x,y]) \leq d$, then $\Delta(x,y) \leq d$.*
2. *If $\mathsf{HAM}(S[x,y], S^R[x,y]) \geq 2d$, then $\Delta(x,y) > (1+\beta) \cdot d$ with probability at least $1 - \frac{1}{n^3}$.*

A position $i \in [x,y]$ is an *isolated mismatch* under p_j if there exists some $r \leq p_j$ for which the subpatterns $S_{r,p_j}[x,y]$ and $S^R_{r,p_j}[x,y]$ differ only in position i. Let $\mathcal{I}_j(x,y)$ be the number of isolated mismatches in $S[x,y]$ under p_j, and let $\mathcal{I}(x,y)$ be the union of $\mathcal{I}_j(x,y)$, over all primes p_j. The next lemma shows that if $\mathsf{HAM}(S[x,y], S^R[x,y]) \leq 2d$, then $\mathcal{I}(x,y)$ is precisely $\mathsf{HAM}(S[x,y], S^R[x,y])$ with high probability over the set of primes.

▶ **Lemma 10.** *(Adaptation of Lemma 4.2 [8]) If $\mathsf{HAM}(S[x,y], S^R[x,y]) \leq 2d$, then $\mathsf{HAM}(S[x,y], S^R[x,y]) = \mathcal{I}(x,y)$ with probability at least $1 - \frac{1}{n^7}$.*

▶ **Lemma 11.** *The set of mismatches can be identified using the second-level fingerprints.*

We are now ready to present the algorithm in full.

NearPalindrome(c_i, x): (determines if $S[c_i, x]$ is a d-near-palindrome)
1. For each $j \in [2\log n]$, initialize $\Delta_j = 0$.
2. For each $j \in [2\log n]$ and $r \in [p_j]$:
   If $\phi^F_{r,p_j}(S[c_i, x]) \neq B^{k+1} \cdot \phi^R_{r,p_j}(S[c_i, x]) \bmod P$, where k is the length of $S_{r,p_j}[c_i, x]$, then increment $\Delta_j(c_i, x) = \Delta_j(c_i, x) + 1$.
3. Let $\Delta(c_i, x) = \max_j\{\Delta_j(c_i, x)\}$.
4. If $\Delta(c_i, x) > (1 + \beta) \cdot d$, then we immediately reject $S[c_i, x]$. (Recall that $\beta = \frac{1}{16}$.)
5. Initialize $\mathcal{I} = \emptyset$.
6. For each mismatch in $S[c_i, x]$, if there exists $q \in \mathcal{Q}$ and such that $\phi^F_{r',q}(S_{r,p}[c_i, x]) \neq B^{k'+1} \cdot \phi^R_{r',q}(S_{r,p}[c_i, x]) \bmod P$, where k' is the length of $S_{r'+rp,pq}[c_i, x]$, for exactly one $r \in [p], r' \in [q]$, then insert the mismatch into $\mathcal{I}(c_i, x)$. (This is the set of *isolated mismatches*.)
7. If $|\mathcal{I}(c_i, x)| > d$, then we reject $S[c_i, x]$.
8. Else, if $|\mathcal{I}(c_i, x)| \leq d$, then we accept $S[c_i, x]$ and return $\mathcal{I}(c_i, x)$.

▶ **Theorem 12.** *With probability at least* $1 - \frac{1}{n^3}$*, procedure* NearPalindrome *returns whether* $S[c_i, x]$ *is a d-near-palindrome.*

Proof of Theorem 12. If $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) > 2d$, then by Lemma 9, $\Delta(c_i, x) > (1 + \beta) \cdot 2d$ with probability at least $1 - \frac{1}{n^3}$ and so NearPalindrome will reject $S[c_i, x]$. Conditioned on $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) \leq 2d$, by Lemma 10 $\mathcal{I}(c_i, x) = \mathsf{HAM}(S[c_i, x], S^R[c_i, x])$ with probability at least $1 - \frac{1}{n^5}$, and so if $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) > d$ the algorithm safely rejects, and otherwise it accepts. Finally, by Lemma 11 the entire set of mismatches $\mathcal{I}(c_i, x)$ can be computed from the second-level subpattern fingerprints. ◀

4.3 Correctness and Space Complexity

In this section, we finish the proof of Theorem 1 by claiming correctness and analyzing the space used by the one-pass streaming algorithm described in Section 4.1. Since we used the spacing of the checkpoints as in [5], we have the following properties.

▶ **Observation 13.** *([5], Observation 16, Lemma 17) At reading* $S[x]$*, for all* $k \geq k_0 = \left\lceil \frac{\log\left(\frac{(1+\alpha)^2}{\alpha}\right)}{\log(1+\alpha)} \right\rceil$*, let* $C_{x,k} = \{c \in \mathcal{C} \mid level\,(c) = k\}$.
1. $C_{x,k} \subseteq [x - 2(1 + \alpha)^k, x]$.
2. *The distance between two consecutive checkpoints of* $C_{x,k}$ *is* $\lfloor \alpha(1 + \alpha)^{k-2} \rfloor$.
3. $|C_{x,k}| = \left\lceil \frac{2(1+\alpha)^k}{\lfloor \alpha(1+\alpha)^{k-2} \rfloor} \right\rceil$.
4. *At any point in the algorithm, the number of checkpoints is* $\mathcal{O}\left(\frac{\log n}{\epsilon \log(1+\epsilon)}\right)$.

▶ **Corollary 14.** *The total space used by the algorithm is* $\mathcal{O}\left(\frac{d \log^7 n}{\epsilon \log(1+\epsilon)}\right)$ *bits. The update time per arriving symbol is also* $\mathcal{O}\left(\frac{d \log^6 n}{\epsilon \log(1+\epsilon)}\right)$.

Proof. The first-level and second-level Karp-Rabin fingerprints consist of integers modulo P for each of the $\mathcal{O}\left(d \log^5 n\right)$ subpatterns. Since $P \in [n^5, n^6]$, then $\mathcal{O}\left(d \log^6 n\right)$ bits of

space are necessary for each fingerprint. Furthermore, by Observation 13, there are $\frac{\log n}{\epsilon \log(1+\epsilon)}$ checkpoints, so the total space used is $\mathcal{O}\left(d \log^7 n\right)$ bits. For each arriving symbol $S[x]$, the algorithm checks possibly the fingerprints of each checkpoint whether the substring is a d-near-palindrome. There are $\mathcal{O}\left(\frac{\log n}{\epsilon \log(1+\epsilon)}\right)$ checkpoints, each with fingerprints of size $\mathcal{O}\left(d \log^5 n\right)$. Each subpattern of a fingerprint may be compared in constant time, so the overall update time is $\mathcal{O}\left(\frac{d \log^6 n}{\epsilon \log(1+\epsilon)}\right)$. ◀

Proof of Theorem 1. Let ℓ_{max} be the length of the longest d-near-palindrome, $S[x, x + \ell_{max} - 1]$, with midpoint m. Let k be the largest integer so that $2(1+\alpha)^{k-1} < \ell_{max}$, where $\alpha = \sqrt{1+\epsilon} - 1$. Let $y = m + (1+\alpha)^{k-1}$ so that $x < y < x + \ell_{max} - 1$. By Observation 13, there exists a checkpoint in the interval $[y - 2(1+\alpha)^{k-1}, y]$. Furthermore, Observation 13 implies consecutive checkpoints of level $k-1$ are separated by distance $\lfloor \alpha(1+\alpha)^{k-2} \rfloor$. Thus, there exists a checkpoint c in the interval $\left[y - 2(1+\alpha)^{k-1}, y - 2(1+\alpha)^{k-1} + \alpha(1+\alpha)^{k-3}\right]$. If procedure NearPalindrome succeeds for this checkpoint on position $m + (m - c)$, then the output $\tilde{\ell}$ of the algorithm is at least

$$2(m-c) \geq 2m - 2y + 4(1+\alpha)^{k-1} - 2\alpha(1+\alpha)^{k-3} = 2(1+\alpha)^{k-1} - 2\alpha(1+\alpha)^{k-3}.$$

Comparing this output with ℓ_{max},

$$\frac{\ell_{max}}{\tilde{\ell}} \leq \frac{2(1+\alpha)^k}{2(1+\alpha)^{k-1} - 2\alpha(1+\alpha)^{k-3}} = \frac{(1+\alpha)^3}{(1+\alpha)^2 - \alpha} \leq (1+\alpha)^2 = 1 + \epsilon.$$

Thus, if procedure NearPalindrome succeeds for all substrings then $\tilde{\ell} \leq \ell_{max} \leq (1+\epsilon)\tilde{\ell}$. Taking Theorem 12 and a simple union bound over all $\mathcal{O}\left(n^2\right)$ possible substrings, procedure NearPalindrome succeeds for all substrings with probability at least $1/n$, and the result follows. ◀

5 Two-Pass *Exact* Streaming Algorithm

In this section, we prove Theorem 3. Namely, we present a two-pass streaming algorithm which returns the longest d-near-palindrome with space $\mathcal{O}\left(d^2 \sqrt{n} \log^6 n\right)$.

Recall that we assume the lengths of d-near-palindromes are even. Thus, for any substring $S[x, y]$ of even length, we define its *midpoint* $m = \left\lfloor \frac{x+y}{2} \right\rfloor$. Upon reading x, we say that $x - \sqrt{n}$ is a candidate midpoint if the sliding window $S[x - 2\sqrt{n}, x]$ is a d-near-palindrome.

First, we modify the one-pass streaming algorithm with additive error in Section 4 so that it returns a list $\mathcal{L}$ of candidate midpoints of d-near-palindromes with length at least $\ell - \frac{\sqrt{n}}{2}$, where ℓ is an estimate of the maximum length output by the algorithm. However, we show in Lemma 16 that the string has a periodic structure which allows us to keep only $\mathcal{O}(d)$ fingerprints in order to recover the fingerprint for any substring between two midpoints.

In the second pass, we explicitly keep the $\frac{\sqrt{n}}{2}$ characters before the starting positions and candidate midpoints of long d-near-palindromes identified in the first pass. We use a procedure Recover to exactly identify the number and locations of mismatches within the d-near-palindromes identified in the first pass. We then use the $\frac{\sqrt{n}}{2}$ characters to extend the near-palindromes until the number of mismatches exceeds $d + 1$.

For an example, see Figure 4.

We first describe a structural property of a series of overlapping d-near-palindromes, showing that they are "almost" periodic.

▶ **Definition 15.** A string S is said to have period π if $S[j] = S[j+\pi]$ for all $j = 1, \ldots, |S'| - \pi$.

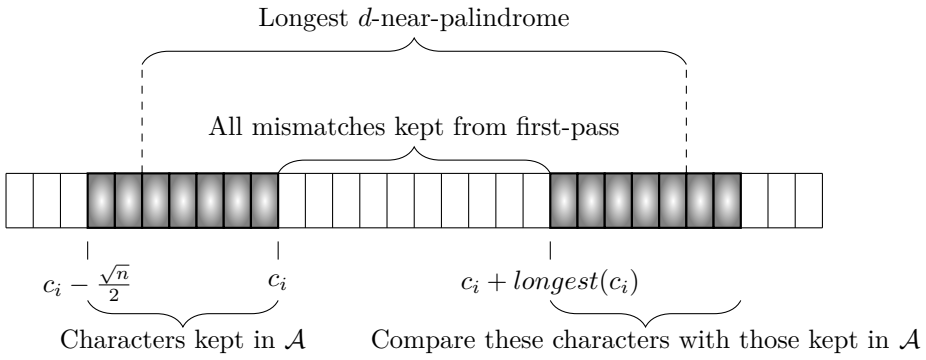


Figure 4 The second pass allows us to find the longest d-near-palindrome by explicitly comparing characters.

The following structural result is a generalization of a structural result about palindromes from [5].

▶ **Lemma 16.** *Let $m_1 < m_2 < \ldots < m_h$ be indices in S that are consecutive midpoints of d-near-palindromes of length ℓ^*, for some integer $\ell^* > 0$. If $m_h - m_1 \leq \ell^*$, then*

1. *$m_1, m_2, \ldots, m_h$ are equally spaced in S', so that $|m_2 - m_1| = |m_i - m_{i+1}|$ for all $i \in [h-1]$.*
2. *There exists string E_h with at most d nonzero entries such that $E_h + S[m_1 + 1, m_h]$ is a prefix of $ww^Rww^R\ldots$ of length at least ℓ^*, for some string w of length $|w| = m_2 - m_1$.*

In the first pass, we specify that the algorithm has sliding window size $2\sqrt{n}$. Thus, if the longest d-near-palindrome has length less than $2\sqrt{n}$, the algorithm can identify it. Otherwise, if the longest d-near-palindrome has length at least $2\sqrt{n}$, then the algorithm finds at most $\frac{\sqrt{n}}{2}$ non-overlapping d-near-palindromes of length at least $\ell - \sqrt{n}$. Hence, $\mathcal{O}\left(d^2\sqrt{n}\log^6 n\right)$ is enough space to store the fingerprints for the substrings between any two candidate midpoints, as well as between checkpoints $s_i \in \mathcal{L}$ and midpoints. This concludes the first pass of the algorithm. Details appear in the [11].

Before we proceed to the second pass, we describe procedure Recover(m_i, m_j, L_c) which either outputs that $S[m_i, m_j]$ is not a d-near-palindrome, or returns the number of mismatches, as well as their indices. The procedure crucially relies on structural result from Lemma 16 to reconstruct the fingerprints of $S[m_i, m_j]$ from fingerprints stored by the first pass. From the reconstructed fingerprints, the subroutine can then determine whether $S[m_i, m_j]$ is a d-near-palindrome, and identify the location of the mismatches, if necessary. The details of procedure Recover appear in [11].

Before the second pass, we first prune the list of checkpoints $\mathcal{C}$ to greedily include only those who are the starting indices for d-near-palindromes of length at least $\tilde{\ell} - \frac{\sqrt{n}}{2}$ and do not overlap with other d-near-palindromes already included in the list. In the second pass, the algorithm keeps track of the $\frac{\sqrt{n}}{2}$ characters before c, for each starting index $c \in \mathcal{C}$. We call procedure Recover to fully recover the mismatches in a region following c. After reading the last symbol in the region, we compare each subsequent symbol with the corresponding symbol before c, counting the total number of mismatches. When the total number of mismatches reaches $d + 1$ after seeing character $S[c + k + j + 1]$, where k is the size of the region, then the previous symbol is the end of the near-palindrome. Hence, the near-palindrome is $S[c - j, c + k + j]$, and if $k + 2j > \tilde{l}$, then we update the information for $\tilde{\ell}$ accordingly. For an example, see Figure 5.

We defer the details of the 2nd pass and the remaining proofs to [11].

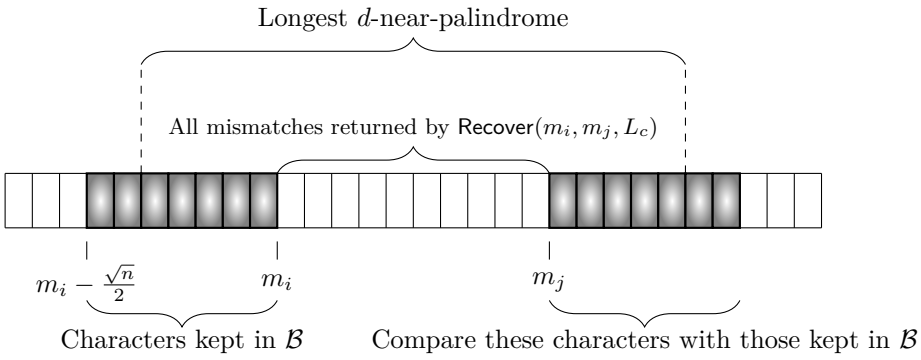


Figure 5 The second pass allows us to find the longest d-near-palindrome by explicitly comparing characters.

Acknowledgments. We would like to thank Funda Ergün, Tatiana Kuznetsova, and Qin Zhang for helpful discussions and pointers.

References

1 List of open problems in sublinear algorithms: Problem 61. URL: `http://sublinear.info/61`.

2 Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

3 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

4 Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments: sketching edit and shift distances. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 931–940, 2013.

5 Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome recognition in the streaming model. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 149–161, 2014.

6 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014.

7 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Inf. Comput.*, 209(4):731–736, 2011.

8 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2039–2052, 2016.

9 Albert A. Conti, Tom Van Court, and Martin C. Herbordt. Processing repetitive sequence structures with mismatches at streaming rate. In *Field Programmable Logic and Application, 14th International Conference , FPL Proceedings*, pages 1080–1083, 2004.

10 Pawel Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw Uznanski. Tight tradeoffs for real-time approximation of longest palindromes in streams. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 18:1–18:13, 2016.

11 Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming for aibohphobes: Longest palindrome with mismatches. *CoRR*, abs/1705.01887, 2017. URL: `http://arxiv.org/abs/1705.01887`.

12 Martin C. Herbordt, Josh Model, Bharat Sukhwani, Yongfeng Gu, and Tom Van Court. Single pass streaming BLAST on fpgas. *Parallel Computing*, 33(10-11):741–756, 2007.

13 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

14 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975.

15 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 315–323, 2009.

16 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM Proceedings*, pages 173–182, 2007.

17 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 222–227, 1977.

Understanding the Correlation Gap for Matchings

Guru Guruganesh[*1] **and Euiwoong Lee**[†2]

1 **Carnegie Mellon University, Pittsburgh, PA, USA**
ggurugan@cs.cmu.edu
2 **Carnegie Mellon University, Pittsburgh, PA, USA**
euiwoonl@cs.cmu.edu

Abstract

Given a set of vertices V with $|V| = n$, a weight vector $w \in (\mathbb{R}^+ \cup \{0\})^{\binom{V}{2}}$, and a probability vector $x \in [0,1]^{\binom{V}{2}}$ in the matching polytope, we study the quantity

$$\frac{\mathbb{E}_G[\nu_w(G)]}{\sum_{(u,v)\in\binom{V}{2}} w_{u,v}x_{u,v}}$$

where G is a random graph where each edge e with weight w_e appears with probability x_e independently, and let $\nu_w(G)$ denotes the weight of the maximum matching of G. This quantity is closely related to correlation gap and contention resolution schemes, which are important tools in the design of approximation algorithms, algorithmic game theory, and stochastic optimization.

We provide lower bounds for the above quantity for general and bipartite graphs, and for weighted and unweighted settings. The best known upper bound is 0.54 by Karp and Sipser, and the best lower bound is 0.4 for bipartite graphs and 0.33 for general graphs. We show that it is at least 0.47 for unweighted bipartite graphs, at least 0.45 for weighted bipartite graphs, and at least 0.43 for weighted general graphs. To achieve our results, we construct local distribution schemes on the dual which may be of independent interest.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G2.2 Graph Theory

Keywords and phrases Matchings, Randomized Algorithms, Contention Resolution Schemes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.32

1 Introduction

We study the size (weight) of the maximum matching of a random graph sampled from various random graph models. Let V be the set of vertices with $|V| = n$. Given the probability vector $x \in [0,1]^{\binom{V}{2}}$ and the weight vector $w \in (\mathbb{R}^+ \cup \{0\})^{\binom{V}{2}}$, let $\mathcal{D}^{\mathsf{G}}_{n,w,x}$ be the distribution of random graphs with n vertices such that each pair $e \in \binom{V}{2}$ becomes an edge with probability x_e independently. If it becomes an edge, its weight is w_e. For bipartite graphs, let V_1 and V_2 be the set of left and right vertices with $|V_1| = |V_2| = n$. Given the probability vector $x \in [0,1]^{V_1 \times V_2}$ and the weight vector $w \in (\mathbb{R}^+ \cup \{0\})^{V_1 \times V_2}$, let $\mathcal{D}^{\mathsf{B}}_{n,w,x}$ be the distribution of random bipartite graphs with $2n$ vertices such that each pair $e \in V_1 \times V_2$ becomes an edge with probability x_e independently. If it becomes an edge, its weight is w_e. We use $\mathcal{D}^{\mathsf{B}}_{n,x}$ (resp. $\mathcal{D}^{\mathsf{G}}_{n,x}$) for the unweighted case ($w = (1,1,\ldots,1)$).

* Supported in part by Anupam Gupta's NSF awards CCF-1319811 and CCF-1536002.
† Supported by a Samsung Fellowship and Venkat Guruswami's NSF CCF-1526092.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 32; pp. 32:1–32:15

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

We focus on the case when the probability vector x is in the matching polytope of the complete (bipartite) graph. Recall that for bipartite graphs, $x \in [0,1]^{V_1 \times V_2}$ is in the matching polytope if each $v \in V_1 \cup V_2$ satisfies $\sum_u x_{u,v} \leqslant 1$. For general graphs, $x \in [0,1]^{\binom{V}{2}}$ is in the matching polytope if each $v \in V$ satisfies $\sum_u x_{u,v} \leqslant 1$ and each odd set $S \subseteq V$ satisfies $\sum_{\{u,v\} \subseteq S} x_{u,v} \leqslant \lfloor (|S|-1)/2 \rfloor$.[1]

Given a weighted graph G, let $\nu_w(G)$ be the weight of the maximum weight matching of G. If G is unweighted, $\nu(G)$ denotes the cardinality of the maximum matching of G. For any $x \in [0,1]^{\binom{V}{2}}$ and $w \in (\mathbb{R}^+ \cup \{0\})^{\binom{V}{2}}$, we have $\mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,x}}[\nu_w(G)] \leqslant \sum_{(u,v) \in \binom{V}{2}} w_{u,v} x_{u,v}$, simply because the probability that (u,v) is included in the maximum matching is at most $x_{u,v}$. The analogous statement also holds for bipartite graphs.

If x is in the matching polytope[2], we can prove that $\mathbb{E}_G[\nu_w(G)] \geqslant \kappa \cdot \sum w_{u,v} x_{u,v}$ for some constant $0 < \kappa < 1$. For the general graph model, κ is known to be at least $(1-1/e)^2 \sim 0.40$ for every w [6]. For the bipartite graph model, κ is known to be at least 0.4 for every w [5]. Karp and Sipser [11] showed an upper bound of 0.54 for both bipartite and general graphs, by demonstrating it for the unweighted models where every edge appears with equal probability. Our main results are the following improved lower bounds on κ. Our first theorem concerns the unweighted bipartite model.

▶ **Theorem 1.1.** *Let $|V_1| = |V_2| = n$ and $x \in [0,1]^{V_1 \times V_2}$ be in the matching polytope of the complete bipartite graph on $V_1 \cup V_2$. Then*

$$\frac{\mathbb{E}_{G \sim \mathcal{D}^{\mathsf{B}}_{n,x}}[\nu(G)]}{\sum_{(u,v) \in V_1 \times V_2} x_{u,v}} \geqslant 0.476. \tag{1}$$

We also obtain a slightly weaker result on the weighted bipartite model.

▶ **Theorem 1.2.** *Let $|V_1| = |V_2| = n$ and $x \in [0,1]^{V_1 \times V_2}$ be in the matching polytope of the complete bipartite graph on $V_1 \cup V_2$. Then for any $w \in (\mathbb{R}^+ \cup \{0\})^{V_1 \times V_2}$,*

$$\frac{\mathbb{E}_{G \sim \mathcal{D}^{\mathsf{B}}_{n,w,x}}[\nu_w(G)]}{\sum_{(u,v) \in V_1 \times V_2} w_{u,v} x_{u,v}} \geqslant \left(1 - \frac{3}{2e}\right) \geqslant 0.4481.$$

Finally, we prove an improved bound on the weighted general graph model.

▶ **Theorem 1.3.** *Let $|V| = n$ and $x \in [0,1]^{\binom{V}{2}}$ be in the matching polytope of the complete graph on $V_1 \cup V_2$. Then for any $w \in (\mathbb{R}^+ \cup \{0\})^{\binom{V}{2}}$,*

$$\frac{\mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,x}}[\nu_w(G)]}{\sum_{(u,v) \in \binom{V}{2}} w_{u,v} x_{u,v}} \geqslant \frac{e^2 - 1}{2e^2} \geqslant 0.4323.$$

1.1 Applications and Related Work

Contention Resolution Schemes and Correlation Gap

Our work is inspired by and related to the rounding algorithms studied in approximation algorithms. Given a downward-closed family $\mathcal{I} \subseteq 2^E$ defined over a ground-set E and a

[1] Our result for general graphs, Theorem 1.3 holds even when x satisfies the first type of constraints.
[2] If x is not in the matching polytope, one can construct examples where $\kappa = \Omega(n)$.

submodular function $f : 2^E \to \mathbb{R}^+$, Chekuri et al. [5] considered the problem of finding $\max_{S\in\mathcal{I}} f(S)$ and introduced *contention resolution schemes* (CR schemes) to obtain improved approximation algorithms for numerous problems. Let $P_{\mathcal{I}}$ be the convex combination of all incidence vectors $\{1_S\}_{S\in\mathcal{I}}$. A c-CR scheme π for $x \in P_{\mathcal{I}}$ is a procedure that, when R is a random subset of E with $e \in R$ independently with probability x_e, returns $\pi(R) \subseteq R$ such that $\pi(R) \in \mathcal{I}$ with probability 1 and $\Pr[e \in \pi(R)] \geqslant c$ for all $e \in E$.

To construct a CR scheme, they introduced the notion of *correlation gap of a polytope*, inspired by [1].[3] Formally, the correlation gap of $\mathcal{I}$ is defined as

$$\kappa(\mathcal{I}) := \inf_{x\in P_{\mathcal{I}},\, w\geqslant 0} \frac{\mathbb{E}_{R\sim\mathcal{D}_x}[\max_{S\subseteq R, S\in\mathcal{I}} \sum_{e\in S} w_e]}{\sum_{e\in E} x_e w_e}, \tag{2}$$

where $\mathcal{D}_x$ is the distribution where each element e appears in R with probability x_e independently. It is easy to see that the existence of c-CR scheme for all $x \in P_{\mathcal{I}}$ implies $\kappa(\mathcal{I}) \geqslant c$. Chekuri et al. [5] proved the converse that every $x \in P_{\mathcal{I}}$ admits a $\kappa(\mathcal{I})$-CR scheme.

By setting E to be the set of all possible edges of a complete (bipartite) graph, and $\mathcal{I}$ to be the set of all matchings of a complete graph, our Theorem 1.2 and Theorem 1.3 for weighted bipartite graphs and weighted general graphs imply that there exist 0.4481-CR scheme and 0.4323-CR scheme for bipartite matching polytopes and general matching polytopes respectively. Note that these lower bounds hold when E' is the set of edges and $\mathcal{I}'$ is a matching polytope of an arbitrary graph G', since

$$\begin{aligned}\kappa(\mathcal{I}) &= \inf_{x\in P_{\mathcal{I}},\, w\geqslant 0} \frac{\mathbb{E}_{R\sim\mathcal{D}_x}[\max_{S\subseteq R, S\in\mathcal{I}} \sum_{e\in S} w_e]}{\sum_{e\in E} x_e w_e} \\ &\leqslant \inf_{x|_{E'}\in P_{\mathcal{I}'},\, w|_{E'}=0} \frac{\mathbb{E}_{R\sim\mathcal{D}_x}[\max_{S\subseteq R, S\in\mathcal{I}} \sum_{e\in S} w_e]}{\sum_{e\in E} x_e w_e} = \kappa(\mathcal{I}').\end{aligned}$$

Maximum Matching of Random Graphs

The study of maximum matchings in random graphs has a long history. It was pioneered by the work of Erdős and Rényi [8, 7], where they proved that a random graph $G_{n,p}$ has a perfect matching with high probability when $p = \Omega(\frac{\ln n}{n})$. The case for sparse graphs was investigated by Karp and Sipser [11] who gave an accurate estimate of $\nu(G)$ for $G_{n,p}$ where $p = \frac{c}{n-1}$ for some constant $c > 0$.

After these two pioneering results, subsequent work has addressed two aspects. The Karp-Sipser algorithm is a simple randomized greedy algorithm, and the first line of works extend the range of models where this algorithm (or its variants) returns an almost maximum matching. Aronson et al. [2] and Chebolu et al. [4] augmented the Karp-Sipser algorithm to achieve tighter results in the standard $G_{n,p}$ model. Bohman and Frieze [3] considered a new model where a graph is drawn uniformly at random from the collection of graphs with a fixed degree sequence and gave a sufficient condition where the Karp-Sipser algorithm finds an almost perfect matching.

The second line of work is based on the following observation: the standard $G_{n,p}$ model, $p = \Omega(\frac{\ln n}{n})$ is required to have a perfect matching, because otherwise there will be an isolated vertex. This naturally led to the question of finding a natural and *sparser* random graph model with a perfect matching. The considered models include a random regular graph, and a $G_{n,p}$ with prescribed minimal degree. We refer the reader to the work of Frieze and Pittel [10] and Frieze [9] and references therein.

[3] [1] defined the correlation gap of a set function $f : 2^E \to \mathbb{R}^+$. Our results apply to this definition too when f denotes the weight of the maximum matching.

1.2 Organization

Our main technical contribution is lower bounding correlation gaps via local distribution schemes for dual variables, which are used to prove Theorem 1.1 and Theorem 1.2 for unweighted and weighted bipartite graphs. We present this framework in Section 2 and prove our bounds for unweighted bipartite graphs (Section 3) and weighted bipartite graphs (Section 4). Our result for weighted general graphs is presented in Section 5.

2 Techinques for Bipartite Graphs

Let $V = V_1 \cup V_2$ be the set of vertices with $|V_1| = |V_2| = n$, $E := V_1 \times V_2$. Fix $w \in (\mathbb{R}^+ \cup \{0\})^E$ and $x \in [0,1]^E$ in the bipartite matching polytope of (V, E).

Our proofs for Theorem 1.1 and 1.2 for bipartite graphs follow the following general framework. Let $G = (V, E(G))$ be a sampled from the distribution where each potential edge $e \in E$ appears with probability x_e independently (recall that $E = V_1 \times V_2$ is the set of all potential edges and $E(G)$ is the edges of one sample G). Let $y(G) \in (\mathbb{R}^+ \cup \{0\})^V$ be an optimal *fractional vertex cover* such that for every $e = (u, v) \in E(G)$, $y_u(G) + y_v(G) \geqslant w_e$. By König-Egerváry theorem, $\|y(G)\|_1 = \nu(G)$.

Given G, consider the situation where initially each vertex v has *mass* $y_v(G)$, and each potential edge has mass $y_e(G) = 0$ (we slightly abuse notation and consider $y(G) \in (\mathbb{R}^+ \cup \{0\})^{V \cup E}$). We construct *local distribution schemes* $F_G : (V \cup E) \times (V \cup E) \to \mathbb{R}$ where $F_G(a, b)$ indicates the amount of mass sent from a to b. We require that $F_G(a, a) = 0$, but we allow $F_G(a, b) \neq -F_G(b, a)$ for $a \neq b$ (the net flow from a to b in this case is $F_G(a, b) - F_G(b, a)$). Let $t(G) \in \mathbb{R}^{V \cup E}$ denote the mass of each vertex and edge after the distribution.

$$t_a(G) := y_a(G) + \sum_{b \in V \cup E} F_G(b, a) - \sum_{b \in V \cup E} F_G(a, b).$$

We choose F_G so that it ensures $t_v(G) \geqslant 0$ for every $v \in V$. This implies

$$\sum_{e \in E} t_e(G) \leqslant \sum_{a \in V \cup E} t_a(G) = \sum_{a \in V \cup E} y_a(G) = \sum_{v \in V} y_v(G) = \nu(G).$$

Therefore, if we prove that for each potential edge $e \in E$

$$\mathbb{E}_G[t_e(G)] \geqslant \alpha \cdot w_e x_e, \tag{3}$$

for some $\alpha > 0$, it implies that

$$\mathbb{E}_G[\nu(G)] \geqslant \alpha \cdot \sum_{e \in E} \mathbb{E}_G[t_e(G)] \geqslant \alpha \cdot \sum_{e \in E} w_e x_e.$$

For weighted and unweighted cases, we construct different local distribution schemes $\{F_G\}_G$ that prove (3) with different values of α.

Weighted Bipartite Graphs

Given a sample $G = (V, E(G))$ and a fractional vertex cover $y \in (\mathbb{R}^+ \cup \{0\})^V$, our $F_G(v, e) = y_v(G)/\deg_G(v)$ if $e \in E(G)$ is an edge incident on $v \in V$, and 0 otherwise. Intuitively, each vertex v *distributes* its mass $y_v(G)$ evenly to its incident edges in G. This clearly satisfies

$t_v(G) \geqslant 0$ for every $v \in V$, and for each $e = (u, v) \in E$, we use the following approximation:

$$\begin{aligned}
\mathbb{E}_G[t_e(G)] &= \Pr[e \in G] \cdot \mathbb{E}_G\left[t_e(G) | e \in G\right] \\
&= x_e \, \mathbb{E}_G\left[\frac{y_u(G)}{\deg_G(u)} + \frac{y_v(G)}{\deg_G(v)} | e \in G\right] \\
&\geqslant x_e \, \mathbb{E}_G\left[(y_u(G) + y_v(G)) \frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G\right] \\
&\geqslant x_e w_e \, \mathbb{E}_G\left[\frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G\right].
\end{aligned}$$

Therefore, to prove Theorem 1.2, it suffices to prove that for every potential edge $e \in E$,

$$\mathbb{E}_{G \sim \mathcal{D}^{\mathsf{B}}_{n,w,x}}\left[\frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G\right] \geqslant 0.4481,$$

when G is sampled from $\mathcal{D}^{\mathsf{B}}_{n,w,x}$ with x in the matching polytope. Experimentally trying several extreme cases indicates that the worst case for $e = (u, v) \in E$ happens when $x_e = \varepsilon$ for very small ε, u has only one other edge e_u with $x_{e_u} = 1 - \varepsilon$, and v is incident on $n - 1$ edges $e_{v_1}, \ldots, e_{v_{n-1}}$ with $x_{e_{v_i}} = \frac{1-\varepsilon}{n-1}$. As ε approaches to 0, $\mathbb{E}_G[\frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G]$ converges to $\mathbb{E}[\frac{1}{1+Y_U}]$ as n grows, where Y_U is drawn from a binomial distribution $B(n-1, \frac{1}{n-1})$. Section 4 formally proves that this is indeed the worst case.

Unweighted Bipartite Graphs

One simple but important observation is that in the above example where $\mathbb{E}_G[t_e(G)] \approx \mathbb{E}[\frac{1}{1+Y_U}]x_e$, e is an edge with very small $x_e = \varepsilon$, and it is adjacent to a large edge e_u with $x_{e_u} = 1 - \varepsilon$. From the persepctive of x_{e_u}, the expected number of adjacent edges is at most 2ε, so $\mathbb{E}_G[t_{e_u}(G)] \approx x_{e_u} \approx 1$. Since e_u gets much more than what it needs ($\mathbb{E}[t_{e_u}] \geqslant 0.476$ suffices to prove Theorem 1.1), it is natural to take some value from $t_{e_u}(G)$ to increase $t_e(G)$.

Formally, given $G = (V, E(G))$, our new local distribution scheme $F_G : (V \cup E) \times (V \cup E) \to \mathbb{R}$ is defined as follows. Let c be an universal constant that will be determined later.

$$F_G(a, b) = \begin{cases} \frac{y_a(G)}{\deg_G(a)} & \text{if } a \in V, b \in E(G), a \in b \\ c x_a^2 x_b & \text{if } a \neq b \in E, a \cap b \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Intuitively, on top of the old local distribution scheme for weighted graphs, each edge e pays $cx_e^2 x_f$ to every adjacent edge f with probability 1 (this quantity does not depend on G). Because this term quadratically depends on the x value of the sender, this payment penalizes edges with large x values to help edges with small x values. For a fixed edge $e = (u, v) \in E$ with fixed $x_e = \varepsilon$, Theorem 3.1 shows that the worst case is when both u and v have $n - 1$ other edges of whose x values are equal to $\frac{1-\varepsilon}{n}$. Finally, Lemma 3.2 shows that $\mathbb{E}[t_e] \geqslant 0.476$ for every $\varepsilon \in (0, 1]$, proving Theorem 1.1.

3 Unweighted Bipartite Graphs

We prove Theorem 1.1 for unweighted bipartite graphs. Given $G = (V, E(G))$, consider the local distribution scheme $F_G : (V \cup E) \times (V \cup E) \to \mathbb{R}$ given in (4). This implies that the

mass after this new distribution scheme for an edge $e = (u, v)$ is given by

$$t_e(G) = \alpha_e(G) + \sum_{f \in E \setminus \{e\} : f \ni u} c(x_e x_f^2 - x_e^2 x_f) + \sum_{g \in E \setminus \{e\} : g \ni v} c(x_g^2 x_e - x_e^2 x_g),$$

where $\alpha_e(G) := y_u(G)/\deg_G(u) + y_v(G)/\deg_G(v)$ denotes the mass after the old distribution scheme used for weighted bipartite graphs. We define $\beta_e(x)$ to be the following.

$$\begin{aligned}\beta_e(x) &:= \mathbb{E}_{G \sim \mathcal{D}^{\mathrm{B}}_{n,x}}[t_e(G)] \\ &= \mathbb{E}_{G \sim \mathcal{D}^{\mathrm{B}}_{n,x}}[\alpha_e(G)] + \sum_{f \in E \setminus \{e\} : f \ni u} c(x_e x_f^2 - x_e^2 x_f) + \sum_{g \in E \setminus \{e\} : g \ni v} c(x_g^2 x_e - x_e^2 x_g)\end{aligned}$$

To prove Theorem 1.1, it suffices to prove that $\beta_e(x) \geqslant 0.476 x_e$ for each e. Fix $e = (u, v)$. Let $e_{u_1}, \ldots, e_{u_{n-1}}$ be $n-1$ other edges incident on u and $e_{v_1}, \ldots, e_{v_{n-1}}$ be $n-1$ other edges incident on v. $\mathbb{E}_{G \sim \mathcal{D}^{\mathrm{B}}_{n,x}}[\alpha_e(G)]$ is lower bounded by $x_e \, \mathbb{E}_G[\frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G]$ as before. Define $F(x_0, y_1, \ldots, y_{n-1}, z_1, \ldots, z_{n-1})$ by

$$\begin{aligned}F(x_0, y_1, \ldots, y_{n-1}, z_1, \ldots, z_{n-1}) := x_0 \, \mathbb{E}[\frac{1}{1 + \max(Y, Z)}] &+ \sum_{i=1}^{n-1} c(x_0 y_i^2 - x_0^2 y_i) \\ &+ \sum_{i=1}^{n-1} c(x_0 z_i^2 - x_0^2 z_i),\end{aligned}$$

where $Y := Y_1 + \cdots + Y_{n-1}$ and $Z := Z_1 + \cdots + Z_{n-1}$ and each Y_i (resp. Z_i) is an independent Bernoulli random variable with $\mathbb{E}[Y_i] = y_i$ (resp. $\mathbb{E}[Z_i] = z_i$). By construction, $\beta_e(x) \geqslant F(x_e, x_{e_{u_1}}, \ldots, x_{e_{u_{n-1}}}, x_{e_{v_1}}, \ldots, x_{e_{v_{n-1}}})$. Given fixed $\sum_{i=1}^{n-1} x_{e_{u_i}}$ and $\sum_{i=1}^{n-1} x_{v_{u_i}}$, the following theorem shows that F is minimized when $x_{e_{u_1}} = \cdots = x_{e_{u_{n-1}}}$ and $x_{e_{v_1}} = \cdots = x_{e_{v_{n-1}}}$.

▶ **Theorem 3.1.** *For $x_0, y_1, \ldots, y_m, z_1, \ldots, z_m \in [0, 1]$ where $y_s := \sum_{i=1}^m y_i \leqslant 1 - x_0$ and $z_s := \sum_{i=1}^m z_i \leqslant 1 - x_0$,*

$$F(x_0, y_1, \ldots, y_m, z_1, \ldots, z_m) \geqslant F(x_0, \frac{y_s}{m}, \ldots, \frac{y_s}{m}, \frac{z_s}{m}, \ldots, \frac{z_s}{m}).$$

Proof. Without loss of generality, assume $y_1 \geqslant \ldots \geqslant y_m$. We will show that if $y_1 > y_m$,

$$\frac{\partial F}{\partial y_m} - \frac{\partial F}{\partial y_1} \leqslant 0. \tag{5}$$

This implies that as long as $y_1 > y_m$, decreasing y_1 and increasing y_m by the same amount will never increase F while maintaining $y_1 + \cdots + y_m = y_s$, so F is minimized when $y_1 = \cdots = y_m = \frac{y_s}{m}$. The same argument for $z_1, \ldots, z_m$ will prove the theorem.

Let $Y := Y_1 + \cdots + Y_m$ and $Z := Z_1 + \cdots + Z_m$, where each Y_i (resp. Z_i) is an independent Bernoulli random variable with $\mathbb{E}[Y_i] = y_i$ (resp. $\mathbb{E}[Z_i] = z_i$). To prove (5), we first compute $\frac{\partial \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}]}{\partial y_m} - \frac{\partial \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}]}{\partial y_1}$. Let $Y' := Y_2 + \cdots + Y_{m-1}$. We decompose $\mathbb{E}[\frac{1}{1+\max(Y,Z)}]$ as follows.

$$
\begin{aligned}
&\mathbb{E}[\frac{1}{1+\max(Y,Z)}] \\
=&\sum_{i=0}^{m}\sum_{j=0}^{m}\left(\Pr[Y=i]\cdot\Pr[Z=j]\cdot\frac{1}{1+\max(i,j)}\right) \\
=&\sum_{i=0}^{m}\left(\Pr[Y'=i]\cdot\Pr[Z\leqslant i](\frac{(1-y_1)(1-y_m)}{1+i}+\frac{y_1(1-y_m)+(1-y_1)y_m}{2+i}+\frac{y_1y_m}{3+i})\right) \\
+&\sum_{i=0}^{m}\left(\Pr[Y'=i]\cdot\Pr[Z=i+1](\frac{1-y_1y_m}{2+i}+\frac{y_1y_m}{3+i})\right) \\
+&\sum_{i=0}^{m}\Pr[Y'=i]\cdot\Pr[Z\geqslant i+2]\cdot\frac{1}{3+i}
\end{aligned}
$$

Therefore, the directional derivative can be written as

$$
\begin{aligned}
&(\frac{\partial}{\partial y_m}-\frac{\partial}{\partial y_1})\,\mathbb{E}[\frac{1}{1+\max(Y,Z)}] \\
=&(y_1-y_m)\sum_{i=0}^{m}\left(\Pr[Y'=i]\cdot\Pr[Z\leqslant i](\frac{1}{1+i}-\frac{2}{2+i}+\frac{1}{3+i})\right) \\
+&(y_1-y_m)\sum_{i=0}^{m}\left(\Pr[Y'=i]\cdot\Pr[Z=i+1](-\frac{1}{2+i}+\frac{1}{3+i})\right) \\
\leqslant&(y_1-y_m)\sum_{i=0}^{m}\left(\Pr[Y'=i]\cdot\Pr[Z\leqslant i](\frac{1}{1+i}-\frac{2}{2+i}+\frac{1}{3+i})\right) \\
\leqslant&(y_1-y_m)\sum_{i=0}^{m}\left(\Pr[Y'=i]\cdot\Pr[Z\leqslant i](\frac{1}{1+i}-\frac{2}{2+i}+\frac{1}{3+i})\right) \\
\leqslant&\frac{y_1-y_m}{3},
\end{aligned}
$$

where the last inequality follows from the fact that

$$
(\frac{1}{1+i}-\frac{2}{2+i}+\frac{1}{3+i})=\frac{2}{(1+i)(2+i)(3+i)}\leqslant\frac{1}{3}.
$$

Finally,

$$
\begin{aligned}
&(\frac{\partial}{\partial y_m}-\frac{\partial}{\partial y_1})F \\
=&(\frac{\partial}{\partial y_m}-\frac{\partial}{\partial y_1})(x_e\,\mathbb{E}[\frac{1}{1+\max(Y,Z)}]+cx_ey_1^2-cx_e^2y_1+cx_ey_m^2-cx_e^2y_m) \\
\leqslant&\frac{x_e(y_1-y_m)}{3}-2cx_e(y_1-y_m)=0.
\end{aligned}
$$

By taking $c=\frac{1}{6}$. ◀

Therefore, for any $e\in E$, $\beta_e(x)\geqslant F(x_e,\frac{y_s}{n-1},\ldots,\frac{y_s}{n-1},\frac{z_s}{n-1},\ldots,\frac{z_s}{n-1})$ for some $y_s\leqslant 1-x_e$ and $z_s\leqslant 1-x_e$. Let

$$\begin{aligned}
G(x_e, y_s, z_s) :=& F(x_e, \frac{y_s}{n-1}, \dots, \frac{y_s}{n-1}, \frac{z_s}{n-1}, \dots, \frac{z_s}{n-1}) \\
=& x_e \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}] + (n-1)c(x_e(\frac{y_s}{n-1})^2 - x_e^2(\frac{y_s}{n-1})) \\
&+ (n-1)c(x_e(\frac{z_s}{n-1})^2 - x_e^2(\frac{z_s}{n-1})) \\
=& x_e \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}] + cx_e y_s((\frac{y_s}{n-1}) - x_e) + cx_e z_s((\frac{z_s}{n-1}) - x_e) \\
\geqslant& x_e \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}] - 2cx_e^2
\end{aligned}$$

where $Y \sim \mathsf{Binomial}(n-1, \frac{y_s}{n-1})$, $Z \sim \mathsf{Binomial}(n-1, \frac{z_s}{n-1})$. Note that the final quantity is minimized when $y_s = z_s = 1 - x_e$. Finally, let

$$H_{n-1}(x_e) := x_e \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}] - 2cx_e^2,$$

where $Y, Z \sim \mathsf{Binomial}(n-1, \frac{1-x_e}{n-1})$.

▶ **Lemma 3.2.** *For any $m \in \mathbb{N}$ and $x_e \in [0,1]$, $H_m(x_e) \geqslant 0.476 x_e$.*

Proof. Since the binomial distribution is approximated by the Poisson distribution in the limit, we use this to ease the calculation. Let $Y, Z \sim \mathsf{Poisson}(1-x)$. Let $H(x) := x \, \mathbb{E}[\frac{1}{1+\max(Y,Z)}] - x^2/3$ (we substitute $c = 1/6$ into the earlier equation). In particular, we write the expectation in full to get

$$\begin{aligned}
\mathbb{E}[\frac{1}{1+\max(Y,Z)}] &= \sum_{k=0}^{\infty}\sum_{j=0}^{\infty} \frac{1}{1+\max(j,k)} e^{-2(1-x)} \frac{(1-x)^{j+k}}{j!k!} \\
&= \frac{1}{e^{2(1-x)}} \sum_{k=0}^{\infty} \Big(\sum_{j=0}^{k} \frac{1}{1+\max(j,k-j)} \frac{1}{j!(k-j)!} \Big) (1-x)^k
\end{aligned}$$

Let $P_t(x)$ denote the above sum truncated at $k = t$. I.e.

$$P_t(x) := \frac{1}{e^{2(1-x)}} \sum_{k=0}^{t} \Big(\sum_{j=0}^{k} \frac{1}{1+\max(j,k-j)} \frac{1}{j!(k-j)!} \Big) (1-x)^k$$

This is a degree t-polynomial in $(1-x)$ with a normalizing factor of $e^{-2(1-x)}$ and note that $\mathbb{E}[\frac{1}{1+\max(Y,Z)}] \geqslant P_t(x)$ for any $t \in \mathbb{N}$.

Truncating this polynomial with $t = 15$, we can see that this has a minimum value of 0.476 for all values of $x \in [0,1]$. we can see that $\mathbb{E}[\frac{1}{1+\max(Y,Z)}] - x/3 \geqslant P_15(x) - x/3$. In the interval $x \in [0,1]$, this function achieves its minimum at $x = 0$ achieving a minimum of 0.476. ◀

4 Weighted Bipartite Graphs

We prove Theorem 1.2 for weighted bipartite graphs. As explained in Section 2, it suffices to prove that for each $e = (u,v) \in E$,

$$\mathbb{E}_{G \sim \mathcal{D}^{\mathsf{B}}_{n,w,x}} \left[\frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G \right] \geqslant 0.4481.$$

Fix $e = (u, v)$ and assume $V = \{v, v_1, \ldots, v_{n-1}\} \cup \{u, u_1, \ldots, u_{n-1}\}$. Let $Y = \deg_G(u) - 1$ and $Z = \deg_G(v) - 1$. Given $e \in G$, Y and Z can be represented as $Y = \sum_{i=1}^{n-1} Y_i$ and $Z = \sum_{i=1}^{n-1} Z_i$, where Y_i indicates where $(u, v_i) \in E(G)$ and Z_i indicates where $(v, u_i) \in E(G)$. This construction ensures that

$$\mathbb{E}_G\left[\frac{1}{\max(\deg_G(u), \deg_G(v))} | e \in G\right] = \mathbb{E}_{Y,Z}\left[\frac{1}{1 + \max(Y, Z)}\right].$$

Note that $Y_1, \ldots, Y_{n-1}, Z_1, \ldots, Z_{n-1}$ are mutually independent, and $\mathbb{E}[Y], \mathbb{E}[Z] \leqslant 1$. By monotonicity, assuming $\mathbb{E}[Y] = \mathbb{E}[Z] = 1$ never increases the lower bound. The following theorem shows that the worst case happens when one of Y, Z is consistently 1 and the other is drawn from $\mathsf{Binomial}(n-1, \frac{1}{n-1})$.

▶ **Theorem 4.1.** *Let $Y = Y_1 + \cdots + Y_m$ and $Z = Z_1 + \cdots + Z_m$, where $Y_1, \ldots, Y_m, Z_1, \ldots, Z_m$ are mutually independent Bernoulli random variables with $\mathbb{E}[Y] = \mathbb{E}[Z] = 1$. Then,*

$$\mathbb{E}\left[\frac{1}{1 + \max(Y, Z)}\right] \geqslant \mathbb{E}\left[\frac{1}{1 + Y_U}\right],$$

where Y_U is drawn from $\mathsf{Binomial}(m, \frac{1}{m})$.

Proof. We decompose $\mathbb{E}[\frac{1}{1+\max(Y,Z)}]$ as follows.

$$\begin{aligned}
&\mathbb{E}\left[\frac{1}{1 + \max(Y, Z)}\right] \\
&= \sum_{i=0}^{m}\sum_{j=0}^{m}\left(\Pr[Y = i] \cdot \Pr[Z = j] \cdot \frac{1}{1 + \max(i, j)}\right) \\
&= \sum_{i=0}^{m} \Pr[Y = i]\left[\left(\sum_{j=0}^{i} \Pr[Z = j]\right) \cdot \frac{1}{1+i} + \left(\sum_{j=i+1}^{m} \Pr[Z = j] \cdot \frac{1}{1+j}\right)\right] \\
&= \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \sum_{i=0}^{m} \Pr[Y = i]\left[\sum_{j=i+1}^{m} \Pr[Z = j]\left(\frac{1}{1+i} - \frac{1}{1+j}\right)\right] \\
&= \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \sum_{j=1}^{m} \Pr[Z = j]\left[\sum_{i=0}^{j-1} \Pr[Y = i]\left(\frac{1}{1+i} - \frac{1}{1+j}\right)\right].
\end{aligned}$$

Let $t_j := \sum_{i=0}^{j-1} \Pr[Y = i] \cdot \left(\frac{1}{1+i} - \frac{1}{1+j}\right)$. We prove the following facts about t_j's.

▶ **Lemma 4.2.** *For all $j \geqslant 3$, $\frac{t_2}{2} \geqslant \frac{t_j}{j}$.*

Proof. Fix $j \geqslant 3$. By the definition of t_2 and t_j,

$$\begin{aligned}
&\frac{t_2}{2} - \frac{t_j}{j} \\
&= \frac{1}{2}\Big(\Pr[Y=0](1-\frac{1}{3}) + \Pr[Y=1](\frac{1}{2}-\frac{1}{3})\Big) - \frac{1}{j}\Big(\sum_{i=0}^{j-1}\Pr[Y=i](\frac{1}{1+i}-\frac{1}{1+j})\Big) \\
&= \frac{1}{3}\Pr[Y=0] + \frac{1}{12}\Pr[Y=1] - \frac{1}{j}\Big(\sum_{i=0}^{j-1}\Pr[Y=i](\frac{1}{1+i}-\frac{1}{1+j})\Big) \\
&= (\frac{1}{3}-\frac{1}{1+j})\Pr[Y=0] + (\frac{1}{12}-\frac{j-1}{2j(j+2)})\Pr[Y=1] \\
&\quad - \frac{1}{j}\Big(\sum_{i=2}^{j-1}\Pr[Y=i](\frac{1}{1+i}-\frac{1}{1+j})\Big) \\
&\geqslant \left(\frac{1}{3}-\frac{1}{1+j}-\frac{1}{j}\sum_{i=2}^{j-1}(\frac{1}{1+i}-\frac{1}{1+j})\right)\Pr[Y=0] + (\frac{1}{12}-\frac{j-1}{2j(j+2)})\Pr[Y=1],
\end{aligned}$$

where the inequality follows from $\Pr[Y=0] \geqslant \Pr[Y=i]$ for $i \geqslant 2$. To prove $\frac{t_2}{2} - \frac{t_j}{j} \geqslant 0$, it suffices to prove that $\frac{1}{3} - \frac{1}{1+j} - \frac{1}{j}\sum_{i=2}^{j-1}(\frac{1}{1+i}-\frac{1}{1+j}) \geqslant 0$, and $\frac{1}{12} - \frac{j-1}{2j(j+2)} \geqslant 0$. It is easy to verify the latter for $j \geqslant 3$. The former can be proved as

$$\begin{aligned}
&\frac{1}{3} - \frac{1}{1+j} - \frac{1}{j}\sum_{i=2}^{j-1}(\frac{1}{1+i}-\frac{1}{1+j}) \\
=&\frac{1}{3} + \frac{j-2}{j(1+j)} - (\frac{1}{1+j} + \frac{1}{j}\sum_{i=2}^{j-1}\frac{1}{1+i}) \\
\geqslant&\frac{1}{3} + \frac{j-2}{j(1+j)} - (\frac{1}{1+j} + \frac{j-2}{3j}) \\
=&(\frac{1}{3} - \frac{j-2}{3j}) + (\frac{j-2}{j(1+j)} - \frac{1}{1+j}) \\
=&\frac{2}{3j} - \frac{2}{j(1+j)} \geqslant 0,
\end{aligned}$$

where the first inequality follows from $\frac{1}{1+i} \leqslant \frac{1}{3}$ for $i \geqslant 2$ and the last inequality follows from $j \geqslant 3$. ◀

We prove the theorem by considering the following two cases.

Case 1: $2\Pr[Y=0] \geqslant \Pr[Y=1]$ **or** $2\Pr[Z=0] \geqslant \Pr[Z=1]$

Without loss of generality, assume that $2\Pr[Y=0] \geqslant \Pr[Y=1]$. It is equivalent to

$$\begin{aligned}
&\Pr[Y=0] \geqslant \frac{2}{3}\Pr[Y=0] + \frac{1}{6}\Pr[Y=1] \\
\Leftrightarrow\quad &t_1 \geqslant \frac{t_2}{2}.
\end{aligned}$$

By Lemma 4.2, it implies that $t_1 \geqslant \frac{t_j}{j}$ for all $j \geqslant 2$. Then, since $\mathbb{E}[Z] = \sum_{j=1}^{m} j \cdot \Pr[Z = j] = 1$,

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{1+\max(Y,Z)}\right] &= \sum_{i=0}^{m} \Pr[Y=i] \cdot \frac{1}{1+i} - \sum_{j=1}^{m} \Pr[Z=j] t_j \\
&\geqslant \sum_{i=0}^{m} \Pr[Y=i] \cdot \frac{1}{1+i} - t_1 \sum_{j=1}^{m} j \cdot \Pr[Z=j] \\
&= \sum_{i=0}^{m} \Pr[Y=i] \cdot \frac{1}{1+i} - t_1 \\
&= \mathbb{E}[\frac{1}{1+\max(Y,1)}].
\end{aligned}$$

The following lemma proves the theorem in the case $t_1 \geqslant \frac{t_2}{2}$.

▶ **Lemma 4.3.** $\mathbb{E}[\frac{1}{1+\max(Y,1)}] \geqslant \mathbb{E}[\frac{1}{1+\max(Y_U,1)}]$.

Proof. Note that $Y = Y_1 + \cdots + Y_m$, and each Y_i is a Bernoulli random variable. Let $y_i := \mathbb{E}[Y_i]$. Without loss of generality, assume $y_1 \geqslant \ldots \geqslant y_m$. We will show that if $y_1 > y_m$,

$$\frac{\partial\, \mathbb{E}[\frac{1}{1+\max(Y,1)}]}{\partial y_m} - \frac{\partial\, \mathbb{E}[\frac{1}{1+\max(Y,1)}]}{\partial y_1} \leqslant 0. \tag{6}$$

This implies that as long as $y_1 > y_m$, decreasing y_1 and increasing y_m by the same amount will never increase $\mathbb{E}[\frac{1}{1+\max(Y,1)}]$ while maintaining $y_1 + \cdots + y_m = 1$, so the expectation is minimized when $y_1 = \cdots = y_m$, or $Y = Y_U$. Consider the following decomposition of $\mathbb{E}[\frac{1}{1+\max(X,Y)}]$.

$$\begin{aligned}
\mathbb{E}_Y\left[\frac{1}{1+\max(1,Y)}\right] &= \Pr[Y=0] \cdot \frac{1}{2} + \sum_{i=1}^{m} \Pr[Y=i] \cdot \frac{1}{1+i} \\
&= \frac{1}{2}(1 - \sum_{i=1}^{m} \Pr[Y=i]) + \sum_{i=1}^{m} \Pr[Y=i] \cdot \frac{1}{1+i} \\
&= \frac{1}{2} - \sum_{i=2}^{m} \Pr[Y=i] \cdot (\frac{1}{2} - \frac{1}{1+i}) \\
&= \frac{1}{2} - \sum_{i=2}^{m} \Pr[Y \geqslant i] \cdot (\frac{1}{i} - \frac{1}{1+i}).
\end{aligned}$$

To prove (6), it suffices to prove that for all $i \geqslant 2$,

$$\frac{\partial \Pr[Y \geqslant i]}{\partial y_m} - \frac{\partial \Pr[Y \geqslant i]}{\partial y_1} \geqslant 0.$$

Let $Y' = Y_2 + \cdots + Y_{m-1}$, and fix $i \geqslant 3$.

$$\begin{aligned}
\Pr[Y \geqslant i] &= \Pr[Y' = i-2] y_1 y_m + \Pr[Y' = i-1]\big(y_1(1-y_m) + (1-y_1)y_m + y_1 y_m\big) \\
&\quad + \Pr[Y' \geqslant i] \\
\frac{\partial \Pr[Y \geqslant i]}{\partial y_1} &= \Pr[Y' = i-2] y_m + \Pr[Y' = i-1]\big(1 - y_m\big)
\end{aligned}$$

Therefore,

$$\frac{\partial \Pr[Y \geqslant i]}{\partial y_m} - \frac{\partial \Pr[Y \geqslant i]}{\partial y_1} = \Pr[Y' = i-2](y_1 - y_m) + \Pr[Y' = i-1]\big(y_m - y_1\big)$$
$$= (y_1 - y_m)\big(\Pr[Y' = i-2] + \Pr[Y' = i-1]\big).$$

Finally, it remains to show that $\Pr[Y' = j] \geqslant \Pr[Y' = j+1]$ for all $j \geqslant 0$. The case $j = 0$ is true since $\Pr[Y' = 0] = \prod_{k=2}^{m-1}(1 - y_k)$ and

$$\Pr[Y' = 1] = \sum_{k=2}^{m-1} \Pr[Y' = 0] \cdot \frac{y_k}{1 - y_k} \leqslant \sum_{k=2}^{m-1} \Pr[Y' = 0] \frac{y_k}{1 - y_2}$$
$$= \frac{\Pr[Y' = 0]}{1 - y_2} \sum_{i=2}^{m-1} y_k \leqslant \Pr[Y' = 0],$$

where the last line follows from $\sum_{k=2}^{m-1} y_i \leqslant 1 - y_1 \leqslant 1 - y_2$ since y_1 is the biggest element. The case $j \geqslant 1$ follows from the fact the sequence $(\Pr[Y' = j])_j$ has one mode or two consecutive modes, and at least one of them occurs at $j = 0$ ($\mathbb{E}[Y'] < 1$ implies $\Pr[Y' = 0] > \Pr[Y' = j]$ for all $j \geqslant 2$). ◀

Case 2: $2\Pr[Y = 0] \leqslant \Pr[Y = 1]$ **and** $2\Pr[Z = 0] \leqslant \Pr[Z = 1]$

Since $\sum_{i=0}^{m} \Pr[Z = i] = 1$ and $\mathbb{E}[Z] = \sum_{i=1}^{m} i \cdot \Pr[Z = i] = 1$, we have $\Pr[Z = 0] = \sum_{i=2}^{m}(i-1)\Pr[Z = i]$. Together with the fact $2\Pr[Z = 0] \leqslant \Pr[Z = 1]$, it implies

$$1 - \Pr[Z = 1] = \Pr[Z = 0] + \sum_{i=2}^{m} \Pr[Z = i] \leqslant 2\Pr[Z = 0] < \Pr[Z = 1],$$

so $\Pr[Z = 1] \geqslant \frac{1}{2}$. Finally,

$$\mathbb{E}\left[\frac{1}{1 + \max(Y, Z)}\right] = \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \sum_{j=1}^{m} \Pr[Z = j] \cdot t_j$$
$$= \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \Pr[Z = 1] \cdot t_1 - \sum_{j=2}^{m} \Pr[Z = j] \cdot t_j$$
$$\geqslant \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \Pr[Z = 1] \cdot t_1 - \sum_{j=2}^{m} j \cdot \Pr[Z = j] \cdot \frac{t_2}{2}$$
$$= \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \Pr[Z = 1] \cdot t_1 - \frac{t_2}{2}(1 - \Pr[Z = 1])$$
$$\geqslant \sum_{i=0}^{m} \Pr[Y = i] \cdot \frac{1}{1+i} - \frac{t_1}{2} - \frac{t_2}{4} = \mathbb{E}\left[\frac{1}{1 + \max(Y, Y_H)}\right],$$

where Y_H is drawn from $\mathsf{Binomial}(2, \frac{1}{2})$. The first inequality follows from Lemma 4.2, and the second inequality follows from $\Pr[Z = 1] \geqslant 0.5$ and $t_1 \leqslant \frac{t_2}{2}$.

Since Y_H satisfies $2\Pr[Y_H = 0] = \Pr[Y_H = 1]$, the analysis for Case 1 shows that $\mathbb{E}[\frac{1}{1+\max(Y,Y_H)}] \geqslant \mathbb{E}[\frac{1}{1+\max(1,Y_U)}]$. ◀

The following lemma finishes the proof of Theorem 1.2.

▶ **Lemma 4.4.** *For any $m \in \mathbb{N}$, if $Y \sim \mathsf{Binomial}(m, \frac{1}{m})$,*

$$\mathbb{E}\left[\frac{1}{1+\max(1,Y)}\right] \geqslant 0.4481$$

Proof. Since the binomial distribution is approximated by the Poisson distribution in the limit, we use this to ease the calculation. Let $Y \sim \mathsf{Poisson}(1)$.

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{1+\max(1,Y)}\right] &= \sum_{k=2}^{\infty} \frac{1}{k+1} \Pr[Y' = k] + \frac{1}{2}\Pr[Y < 2] \\
&= \sum_{k=2}^{\infty} \frac{1}{k+1}\frac{1}{k! \cdot e} + \frac{1}{2}(\frac{1}{e} + \frac{1}{e}) \\
&= \frac{1}{e}(\sum_{k=0}^{\infty} \frac{1}{k!} - 1 - 1 - \frac{1}{2}) + \frac{1}{2}(\frac{1}{e} + \frac{1}{e}) \\
&= (e - \frac{5}{2})\frac{1}{e} + \frac{1}{e} \\
&\geqslant 0.4481
\end{aligned}$$

◀

5 General Graphs

In this section, we prove Theorem 1.3 for weighted general graphs. Our proof methods here closely follow that of Lemma 4.9 of Chekuri et al. [5] that lower bounds the correlation gap for monotone submodular functions by $1 - 1/e$. The only difference is that Lemma 5.1 holds for matching with a weaker guarantee (if ν was a monotone submodular function, Lemma 5.1 would hold with $2\nu(G)$ replaced by $\nu(G)$).

Proof. Fix weights $w \in (\mathbb{R}^+ \cup \{0\})^E$. Define $F : [0,1] \to (\mathbb{R}^+ \cup \{0\})$ as $F(x) := \mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,x}}[\nu(G)]$. Now, fix $x \in [0,1]^E$ in the matching polytope. We will show $F(x) \geqslant 0.43 \sum_{e \in E} w_e x_e$.

Consider the function $\phi(t) := F(tx)$ for $t \in [0,1]$.

$$\frac{d\phi}{dt} = x \cdot \nabla F(tx) = \sum_{e \in E} x_e \frac{\partial F}{\partial x_e}\bigg|_{tx} \tag{7}$$

For each $e \in E$,

$$\begin{aligned}
\frac{\partial F}{\partial x_e}\bigg|_{tx} &= \frac{\partial\, \mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,tx}}[\nu(G)]}{\partial x_e}\bigg|_{tx} \\
&= \mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,tx}}[\nu(G) | e \in G] - \mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,tx}}[\nu(G) | e \notin G] \\
&= \mathbb{E}_{G \sim \mathcal{D}^{\mathsf{G}}_{n,w,tx}}[\nu(G \cup \{e\}) - \nu(G \setminus \{e\})],
\end{aligned}$$

where $G \cup \{e\}$ (resp. $G \setminus \{e\}$) denotes the graph $(V, E(G) \cup \{e\})$ (resp. $(V, E(G) \setminus \{e\})$).

▶ **Lemma 5.1.** *For any fixed graph G with weights $\{w_e\}$ and any point x in the matching polytope,*

$$\sum_{e \in E} x_e\big(\nu(G \cup \{e\}) - \nu(G \setminus \{e\})\big) + 2\nu(G) \geqslant \sum_{e \in E} x_e w_e.$$

Proof. Let $M \subseteq E(G)$ be a maximum weight matching of G. Note that

$$\begin{aligned}
&\sum_{e\in E} x_e\big(\nu(G\cup\{e\}) - \nu(G\setminus\{e\})\big) + 2\nu(G)\\
\geqslant &\sum_{e\in E} x_e\big(\nu(G\cup\{e\}) - \nu(G)\big) + 2\sum_{f\in M} w_f\\
\geqslant &\sum_{e\in E} x_e\big(\nu(G\cup\{e\}) - \nu(G)\big) + \sum_{f\in M}\sum_{e\in E: e\sim f} x_e w_f \qquad (8)
\end{aligned}$$

where $f \sim e$ indicates that two edges f and e share an endpoint. To prove the lemma, it suffices to show that for each $e \in E$, the coefficient of of x_e in (8) is at least w_e. We consider the following cases.

- If $M\cup\{e\}$ is a matching, $\nu(G\cup\{e\}) \geqslant \nu(G) + w_e$ and $\nu(G\setminus\{e\}) \leqslant \nu(G)$, so $\nu(G\cup\{e\}) - \nu(G\setminus\{e\}) \geqslant w_e$.
- If e intersects exactly one edge $f \in M$, the coefficient of x_e is $\nu(G\cup\{e\}) - \nu(G) + w_f$. If $w_f \geqslant w_e$, it is at least w_e. If $w_f < w_e$, $M\cup\{e\}\setminus\{f\}$ is a matching of weight $\nu(G)+w_e-w_f$. It implies that $e \notin E(G)$ and $\nu(G\cup\{e\})-\nu(G) \geqslant w_e-w_f$, so $\nu(G\cup\{e\})-\nu(G)+w_f \geqslant w_e$.
- If e intersects two edges $f, g \in M$, the coefficient of x_e is $\nu(G\cup\{e\}) - \nu(G) + w_f + w_g$. If $w_f + w_g \geqslant w_e$, it is at least w_e. If $w_f + w_g < w_e$, $M\cup\{e\}\setminus\{f,g\}$ is a matching of weight $\nu(G) + w_e - w_f - w_g$. It implies that $e \notin E(G)$ and $\nu(G\cup\{e\}) - \nu(G) \geqslant w_e - w_f - w_g$, so $\nu(G\cup\{e\}) - \nu(G) + w_f + w_g \geqslant w_e$. ◀

Combining (7) and Lemma 5.1,

$$\begin{aligned}
\frac{d\phi}{dt} &= \sum_{e\in E} x_e \frac{\partial F}{\partial x_e}\bigg|_{tx}\\
&= \sum_{e\in E} \mathbb{E}_{G\sim\mathcal{D}^{\mathsf{G}}_{n,w,tx}}[\nu(G\cup e) - \nu(G\setminus e)]\\
&\geqslant \sum_{e\in E} x_e w_e - 2\,\mathbb{E}_{G\sim\mathcal{D}^{\mathsf{G}}_{n,w,tx}}[\nu(G)]\\
&= \sum_{e\in E} x_e w_e - 2\phi(t).
\end{aligned}$$

which implies that,

$$\frac{d}{dt}(e^{2t}\phi(t)) = 2e^{2t}\phi(t) + e^{2t}\frac{d\phi}{dt} \geqslant e^{2t}\sum_{e\in E} x_e w_e.$$

Since $\phi(0) = 0$,

$$e^2\phi(1) \geqslant \sum_{e\in E} x_e w_e \int_0^1 e^{2t}dt = \frac{e^2-1}{2}\sum_{e\in E} x_e w_e,$$

which proves the theorem. ◀

References

1 Shipra Agrawal, Yichuan Ding, Amin Saberi, and Yinyu Ye. Price of correlations in stochastic optimization. *Operations Research*, 60(1):150–162, 2012.

2 Jonathan Aronson, Alan Frieze, and Boris G Pittel. Maximum matchings in sparse random graphs: Karp-sipser revisited. *Random Structures and Algorithms*, 12(2):111–177, 1998.

3 Tom Bohman and Alan Frieze. Karp–sipser on random graphs with a fixed degree sequence. *Combinatorics, Probability and Computing*, 20(05):721–741, 2011.

4 Prasad Chebolu, Alan M. Frieze, and Páll Melsted. Finding a maximum matching in a sparse random graph in $O(n)$ expected time. *J. ACM*, 57(4):24:1–24:27, 2010. doi:10.1145/1734213.1734218.

5 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014. doi:10.1137/110839655.

6 Marek Cygan, Fabrizio Grandoni, and Monaldo Mastrolilli. How to sell hyperedges: the hypermatching assignment problem. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 342–351. Society for Industrial and Applied Mathematics, 2013.

7 P Erdős and A Rényi. On random matrices ii. *Studia Sci. Math. Hungar.*, 3:459–464, 1968.

8 P Erdős and Alfréd Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Mathematica Hungarica*, 17(3-4):359–368, 1966.

9 Alan Frieze. Perfect matchings in random bipartite graphs with minimal degree at least 2. *Random Structures and Algorithms*, 26(3):319–358, 2005.

10 Alan Frieze and Boris Pittel. Perfect matchings in random graphs with prescribed minimal degree. In *Mathematics and Computer Science III*, pages 95–132. Springer, 2004.

11 Richard M Karp and Michael Sipser. Maximum matching in sparse random graphs. In *Foundations of Computer Science, 1981. SFCS'81. 22nd Annual Symposium on*, pages 364–375. IEEE, 1981.

Hardness of Rainbow Coloring Hypergraphs*†

Venkatesan Guruswami[1] and Rishi Saket[2]

1 **School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA**
venkatg@cs.cmu.edu
2 **IBM Research, Bangalore, India**
rissaket@in.ibm.com

Abstract

A hypergraph is *k-rainbow colorable* if there exists a vertex coloring using k colors such that each hyperedge has all the k colors. Unlike usual hypergraph coloring, rainbow coloring becomes *harder* as the number of colors increases. This work studies the rainbow colorability of hypergraphs which are guaranteed to be nearly balanced rainbow colorable. Specifically, we show that for any $Q, k \geq 2$ and $\ell \leq k/2$, given a Qk-uniform hypergraph which admits a k-rainbow coloring satisfying:

- in each hyperedge e, for some $\ell_e \leq \ell$ all but $2\ell_e$ colors occur exactly Q times and the rest $(Q \pm 1)$ times,

it is NP-hard to compute an independent set of $(1 - \frac{\ell+1}{k} + \varepsilon)$-fraction of vertices, for any constant $\varepsilon > 0$. In particular, this implies the hardness of even (k/ℓ)-rainbow coloring such hypergraphs.

The result is based on a novel long code PCP test that ensures the strong balancedness property desired of the k-rainbow coloring in the completeness case. The soundness analysis relies on a mixing bound based on uniform reverse hypercontractivity due to Mossel, Oleszkiewicz, and Sen, which was also used in earlier proofs of the hardness of $\omega(1)$-coloring 2-colorable 4-uniform hypergraphs due to Saket, and k-rainbow colorable $2k$-uniform hypergraphs due to Guruswami and Lee.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Fourier analysis of Boolean functions, hypergraph coloring, Inapproximability, Label Cover, PCP

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.33

1 Introduction

A hypergraph is a collection of *vertices* and subsets of the set of vertices called *hyperedges*. It is q-uniform if each hyperedge has exactly q vertices, in particular a 2-uniform hypergraph is the usual graph. An independent set of a hypergraph is a subset of vertices that does not contain all the vertices of any hyperedge. A fundamental property of a hypergraph is its *colorability*: it is said to be k-colorable if the set of vertices can be colored using k colors so that no hyperedge is monochromatic. These color classes partition the vertices into k disjoint independent sets.

The computational aspects of coloring graphs and hypergraphs have been the focus of a substantial body of research. In brief, it is known that 3-colorable graphs can efficiently be colored using $n^{O(1)}$-colors [25] (see also [39, 6, 7, 24, 20, 1]). Similar $n^{O(1)}$-approximations

* Research of the first author supported in part by NSF grant CCF-1526092. This work was initiated during a visit by the first author to Microsoft Research, Bangalore.
† A full version of the paper is available at [19], https://eccc.weizmann.ac.il/report/2017/147/.

are known for 2-colorable 3-uniform and 4-uniform hypergraphs [11, 26, 32]. From the intractability perspective, on graphs the best lower bound only rules out efficiently coloring a 3-colorable graph using 4-colors [27, 16]. For hypergraphs much better lower bounds rule out efficiently coloring 2-colorable 8-uniform hypergraphs using $\exp((\log n)^{\Omega(1)})$ colors [31, 23, 38] building upon the previous $\exp(\exp(\Omega(\sqrt{\log \log n}))$ lower bound [12, 14]. For 2-colorable 4-uniform hypergraphs a corresponding $(\log n)^c$ lower bound is known [37]. These intractability results proceed by ruling out computing independent sets of density (i.e., fraction of vertices) $\alpha(n)$ which implies the same for $(1/\alpha(n))$-coloring the hypergraph. For 2-colorable 3-uniform hypergraphs however, [13] directly showed the hardness of coloring using constantly many colors.

Another notion of coloring in hypergraphs is that of *rainbow colorability* – a hypergraph can be k-rainbow colored if there exists a coloring of the vertices using k colors ensuring that each hyperedge contains vertices of all the k colors. Any $(k-1)$ of the k color classes therefore constitute an independent set, and thus such hypergraphs have at least one independent set of density $(1-1/k)$. Note that unlike usual hypergraph coloring, rainbow coloring becomes more restrictive as the number of colors increases, and the problem is to determine the largest k for which there exists a k-rainbow coloring. This has been studied by Bollobás et. al. [8] who gave structural upper and lower bounds for several classes of hypergraphs. In some scenarios (such as modelling fair resource allocations) it is desirable that the coloring also be *balanced*, i.e., the colors should occur roughly the same number of times in any hyperedge. This is related to minimizing the *discrepancy* of hypergraph 2-colorings, for which notable recent works [5, 33] have given constructive algorithms. Subsequent works have shown tight hardness results for zero discrepancy case in hypergraphs with unbounded hyperedges [10] and for bounded hypergraphs with nearly zero discrepancy [4].

A perfectly balanced rainbow coloring is one in which every color appears exactly the same number of times in each hyperedge. Such hypergraphs are easy to efficiently 2-color using semi-definite programming (see [18]). In particular, a k-uniform k-rainbow colorable hypergraph (a.k.a., a k-uniform k-partite hypergraph) can efficiently be 2-colored. On the other hand, efficient 2-coloring, or even finding an independent set of density 1/2 does not seem possible if the guaranteed coloring deviates from being perfectly balanced. Indeed, Guruswami and Lee [18] proved the hardness of approximately coloring a class of such hypergraphs. They proved that it is NP-hard to compute an independent set of density ε in a Qk-uniform hypergraph $(Q, k \geq 2)$ which is guaranteed to be k-rainbow colorable such that each color appears at least $(Q-1)$ times in every hyperedge. This implies the hardness of coloring such hypergraphs using constantly many colors as well as that of non-trivially (i.e., using at least 2 colors) rainbow coloring them. The results of [18] do not, however, say anything about coloring k-rainbow colorable q-uniform hypergraphs for $q < 2k$. Brakensiek and Guruswami [9], under a conjectured intractability of a problem called "V Label Cover" that they formulate, proved hardness of finding an independent set of density ε in a $(k+1)$-uniform k-rainbow colorable hypergraph. This generalized the case of 2-colorable 3-uniform hypergraphs for which the same hardness was shown by Khot and Saket [30] under the d-to-1 Games Conjecture of Khot [29]. Unlike these conjecture based works however, our focus is on unconditional results.

While the structural guarantee considered in [18] captures balanced rainbow colorings, it also allows those in which a particular color may appear up to $(k + Q - 1)$ times in a hyperedge. This is quite far off from being balanced in the regime where k is comparable or larger than Q, for example when $Q = 2$ which corresponds to the smallest uniformity relative to k for which the hardness applies. The focus of this work is the case of rainbow colorable

hypergraphs with a stronger balancedness condition on the coloring: in each hyperedge the occurrences of some of the colors is each off by *at most* 1 from Q, and the rest of the colors have precisely Q occurrences. In our main result we show that it is hard to rainbow color such hypergraphs even with far fewer colors. Formally we show the following:

▶ **Theorem 1.** *For any $Q, k \geq 2$, $\ell \leq k/2$ and an arbitrarily small constant $\varepsilon > 0$, given a Qk-uniform hypergraph of size n which is guaranteed to be k-rainbow colorable such that:*

- *in each hyperedge e, for some $\ell_e \leq \ell$, there are ℓ_e colors that occur exactly $(Q-1)$ times, ℓ_e colors that occur exactly $(Q+1)$ times and the rest of the colors occur exactly Q times,*

it is NP*-hard to compute an independent set of density $\left(1 - \frac{(\ell+1)}{k} + \varepsilon\right)$. This implies, in particular, that it is* NP*-hard to (k/ℓ)-rainbow color such hypergraphs. Under* DTIME$(N^{O(\log\log N)})$ *reductions from* 3SAT *one can choose $\varepsilon = (\log n)^{-c}$ where c is some positive constant depending on Q, k and ℓ.*

Our result yields (with $Q = k = 2$ and $\ell = 1$) the result of Saket [37] who showed the currently best hardness of $(\log n)^c$-coloring 2-colorable 4-uniform hypergraphs, improving on the $\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$ lower bound by Guruswami, Håstad, and Sudan [15], and Holmerin [22]. For $\ell = 1$ and $Q, k \geq 2$, Guruswami and Lee [18] studied this problem and claimed[1] a weaker hardness – in an auxiliary result of their initial work – ruling out independent sets of density $(1 - 1/k)$.

1.1 Our Techniques

The hardness reduction presented in this paper follows the template of *long code* based Probabilistically Checkable Proofs (PCPs) for the Label Cover problem. The long code encodings of the supposed labels of the Label Cover variables constitute the proof, whose locations are the vertices of the resulting hypergraph instance. The PCP verifier queries a few locations of the proof in each of its random tests defining the set of hyperedges. The test accepts if the $[k] := \{1, \ldots, k\}$-valued labels at the queried locations describe a rainbow coloring satisfying the desired balancedness criterion.

Let us now illustrate what we consider the novelty of our reduction: the PCP test for proving Theorem 1 with constants $Q, k \geq 2$ and $1 \leq \ell \leq k/2$. The test queries k locations from Q long codes corresponding to Q vertices of the Label Cover instance with constraints projecting on a common neighbor. Its main building block is a distribution $\overline{\mathcal{P}}$ over $\prod_{j=1}^{Q}([k]^d)^k$ which gives the query locations restricted to the Q pre-images of a label of the common neighbor. To construct $\overline{\mathcal{P}}$ we define, for $1 \leq t \leq k$, μ_t to be the uniform distribution over all $(\overline{x}^{(1)}, \ldots, \overline{x}^{(t)}) \in ([k]^d)^t$ such that for every $i \in [d]$, $\overline{x}_i^{(1)}, \ldots, \overline{x}_i^{(t)}$ are distinct. Figure 1a illustrates a sample from μ_k.

Corresponding to the jth long code ($j \in [Q]$), let $(\overline{x}^{1,j}, \ldots, \overline{x}^{k,j})$ be an i.i.d. sample from μ_k. Now, for any choice of labels (having the fixed common projection) given by $i_1, \ldots, i_Q \in [d]$,

$$(\overline{x}_{i_1}^{(1,1)}, \ldots, \overline{x}_{i_1}^{(k,1)}, \ldots, \overline{x}_{i_j}^{(1,j)}, \ldots, \overline{x}_{i_j}^{(k,j)}, \ldots, \overline{x}_{i_Q}^{(1,Q)}, \ldots, \overline{x}_{i_Q}^{(k,Q)}) \in [k]^{kQ}$$

is perfectly balanced i.e., each color in $[k]$ occurs exactly Q times. For the PCP test to work, it requires some perturbation while ensuring that the resultant coloring above remains nearly balanced.

[1] However, Guruswami and Lee have since withdrawn this claim (Theorem 1.4 and Appendix D in the ECCC version [17]) from later versions of their paper [18].

2	4	1	5	3
3	1	2	4	5
5	2	4	1	3
1	2	3	4	5

(a) A sample from μ_k

4	2	1	4	3
3	1	1	2	3
3	5	4	1	5
5	4	5	4	2

(b) A sample from $(\mu_\ell \otimes \mu_{k-\ell})$

Figure 1 Samples from μ_k and $(\mu_\ell \otimes \mu_{k-\ell})$. $k = 5$, $\ell = 2$ and $d = 4$.

For this purpose, we choose $j^* \in [Q]$ at random and sample $(\overline{x}^{1,j^*}, \ldots, \overline{x}^{k,j^*})$ from $\mu_\ell \otimes \mu_{k-\ell}$ (illustrated in Figure 1b). Formally, we define:

$$\overline{\mathcal{P}} := \frac{1}{Q} \sum_{j^*=1}^{Q} \left[\underbrace{\mu_k \otimes \cdots \otimes \mu_k}_{j^*-1 \text{ times}} \otimes (\mu_\ell \otimes \mu_{k-\ell}) \otimes \overbrace{\mu_k \otimes \cdots \otimes \mu_k}^{Q-j^* \text{ times}} \right].$$

Notice that the marginal of $\overline{\mathcal{P}}$ for any $j \in [Q]$ is $\mathcal{P} := (1 - 1/Q)\mu_k + (1/Q)(\mu_\ell \otimes \mu_{k-\ell})$. As desired, for any choice of $i_1, \ldots, i_Q \in [d]$, for any $\prod_{j=1}^{Q}(\overline{x}^{(1,j)}, \ldots, \overline{x}^{(k,j)})$ in the support of $\overline{\mathcal{P}}$,

$$(\overline{x}^{(1,1)}_{i_1}, \ldots, \overline{x}^{(k,1)}_{i_1}, \ldots, \overline{x}^{(1,j)}_{i_j}, \ldots, \overline{x}^{(k,j)}_{i_j}, \ldots, \overline{x}^{(1,Q)}_{i_Q}, \ldots, \overline{x}^{(k,Q)}_{i_Q})$$

is nearly balanced: for some $\ell' \leq \ell$, ℓ' of the colors in $[k]$ occur exactly $(Q-1)$ times, ℓ' of them occur exactly $(Q+1)$ times, and the rest of the colors exactly Q times.

To extend the test distribution to the pre-images of L labels on smaller side of the Label Cover, we sample

$$(x^{(1,1)}, \ldots, x^{(k,1)}, \ldots, x^{(1,j)}, \ldots, x^{(k,j)}, \ldots, x^{(1,Q)}, \ldots, x^{(k,Q)}) \in \prod_{j=1}^{Q} ([k]^{Ld})^k$$

by sampling independently for each $i \in [L]$, the restriction of the above locations to the coordinates $\{d(i-1)+1, \ldots, di\}$ from $\overline{\mathcal{P}}$.

Let $f_j : [k]^{Ld} \to \{0,1\}$ denote the restriction to the jth long code of a sufficiently dense independent set. Our soundness analysis shows that with significant probability over the choice of the Q long codes of the PCP test, two of the functions $\{f_j\}_{j\in[Q]}$ are intersecting juntas. Otherwise, the expectations inside each long code would be uncorrelated yielding a hyperedge inside the supposed independent set. These juntas are then decoded into a good labeling for the Label Cover. This motivates the first step of the analysis: in a single long code lower bounding the expectation $\mathbb{E}\left[\prod_{s=1}^{k} f_j(x^{(s,j)})\right]$ for some sufficiently heavy f_j. We show that when $\mathbb{E}[f_j] \geq (1 - \frac{\ell+1}{k} + \varepsilon)$,

$$\mathbb{E}\left[\prod_{s=1}^{k} f_j(x^{(s,j)})\right] \geq \Omega(\varepsilon^c)$$

for some $c > 0$ depending on Q, k and ℓ. This is proved by representing the product inside the expection as $A(X)B(Y)$ where $(X, Y) = ((x^{(1,j)}, \ldots, x^{(\ell,j)}), (x^{(\ell+1,j)}, \ldots, x^{(k,j)}))$. It can be seen that X and Y are at most $(1-1/Q)$-correlated[2]. Further, using the structure of the

[2] This is analogous to the notion of ρ-correlation used in [35] and was also used in the reverse hypercontractivity based mixing bounds of [18].

PCP test we show that $\mathbb{E}[A], \mathbb{E}[B] \geq \Omega(\varepsilon)$. Using this coupled with a lower bound on the mixing of Markov chains that was shown by Mossel, Oleszkiewicz, and Sen [35] based on their generalized *reverse hypercontractivity*, yields the desired lower bounds. A similar use (for a simpler PCP test) of this technique was made by Saket [37]. Subsequently Guruswami and Lee [18] used reverse hypercontractivity to analyze a PCP test which sampled uniformly from $([k]^d)^k$ instead of $\mu_\ell \otimes \mu_{k-\ell}$ for the j^*th long code, and used it to show their NP-hardness result for $O(1)$-coloring k-rainbow colorable hypergraphs mentioned earlier.

In their work, Guruswami and Lee [18] leveraged a *smoothness* property (first defined by Khot [28]) of the Label Cover instance for their analysis which used Gaussian invariance theorems and decoded a labeling using influential coordinates. Unfortunately, achieving smoothness generates a significant blowup in the size of the Label Cover which renders the reduction somewhat inefficient. In contrast, our analysis (as also in [37]) is based on standard Fourier analysis and uses a projection size preservation property of the vanilla Label Cover shown by Håstad [21]. This limits the size blowup enabling us to upper bound the "error" ε in the NO case to $1/\text{poly}(\log n)$ under quasi-polynomial time reductions.

2 Preliminaries

Consider for $i = 1$ to n, the product space $(\Omega_i^{(1)} \times \Omega_i^{(2)}, \mu_i)$ where the marginals of μ_i are $\mu_i^{(1)}, \mu_i^{(2)}$. Let $(\Omega^{(s)}, \mu^{(s)}) = (\prod_{i=1}^n \Omega_i^{(s)}, \otimes_{i=1}^n \mu_i^{(s)})$, for $s = 1, 2$. We say that $(X, Y) \in \Omega^{(1)} \times \Omega^{(2)}$ is ρ-correlated[3] if independently for each $i \in [n]$, (X_i, Y_i) is sampled from μ_i with probability ρ, and from $\mu_i^{(1)} \otimes \mu_i^{(2)}$ with probability $(1 - \rho)$.

The following theorem is a straightforward generalization of the special case of $\Omega^{(1)} = \Omega^{(2)}$ and $\mu_i = \text{id}$ proved in [35]. The derivation is provided in [18] and we incorporate the explicit parameters from [35].

▶ **Theorem 2.** *In above setup, let $A \subseteq \Omega^{(1)}$ and $B \subseteq \Omega^{(2)}$ be two sets such that $\mu^{(1)}\{A\}$, $\mu^{(2)}\{B\} \geq \delta \geq 0$. Let $(X, Y) \in (\Omega^{(1)} \times \Omega^{(2)})$ be ρ-correlated. Then, $\Pr[X \in A, Y \in B] \geq \delta^{\frac{2-\sqrt{\rho}}{1-\sqrt{\rho}}}$.*

We shall also use the *Efron-Stein* decompositions of functions over product spaces (see [34] for reference).

▶ **Proposition 3.** *Let $(\Omega = \prod_{i=1}^n \Omega_i, \mu = \otimes_{i=1}^n \mu_i)$ be a product space. Then, any $f \in L^2(\Omega, \mu)$ can be decomposed uniquely as:*

$$f(x) = \sum_{S \subseteq [n]} f_S(x),$$

where f_S depends only on the coordinates in S and for $S' \not\supseteq S$, $E[f_S | x_{S'}] = 0$. In particular $\{f_S\}_{S \subseteq [n]}$ are orthogonal i.e., $\mathbb{E}[f_S f_{S'}] = 0$ for $S \neq S'$.

The starting point of the reduction is the LabelCover problem which is defined as follows.

▶ **Definition 4.** An instance $\mathcal{L}$ of LabelCover consists of a bipartite graph $G_{\mathcal{L}}(U, V, E)$ along with label sets $[L]$ and $[M]$ where $M = dL$. For each edge e between $u \in U$ and $v \in V$, there is a projection $\pi_{vu} : [M] \mapsto [L]$, such that $|\pi_{vu}^{-1}(j)| = d$ for each $j \in [L]$. A labeling $l_u \in [L]$ to u and $l_v \in [M]$ to v satisfies the edge if $\pi_{vu}(l_v) = l_u$. The goal is to find a labeling of U and V to satisfy the maximum number of edges.

[3] See Footnote 2.

The inapproximability of LabelCover stated below follows from the PCP Theorem [3, 2], Raz's Parallel Repetition Theorem [36]. We also leverage a structural property proved by Håstad [21] showing that for any vertex in V the image of a large subset of its labels remains large under most of the projections incident on v.

▶ **Theorem 5.** *For every positive integer r, there is a deterministic $N^{O(r)}$ time reduction from a* 3SAT *instance of size N to an instance $\mathcal{L}(G_{\mathcal{L}}(U,V,E),\{\pi_{vu}\}_{\{v,u\}\in E},[L],[M])$ of* LabelCover *with the following properties:*

(a) *$|U|,|V| \le N^{O(r)}$. $L, M, d \le 2^{3r}$. G is bi-regular with left and right degrees bounded by $2^{O(r)}$.*

(b) *There is a universal constant $c_0 > 0$ such that for any $v \in V$ and $S \subseteq [M]$, taking an expectation over a random neighbor u of v, $\mathbb{E}\left[|\pi_{vu}(S)|^{-1}\right] \le |S|^{-2c_0}$. This implies that over the choice of a random neighbor u of v,*

$$\Pr\left[|\pi_{vu}(S)| < |S|^{c_0}\right] \le |S|^{-c_0}.$$

(c) *There is a universal constant $\gamma_0 > 0$ such that,*

YES Case: *If the* 3SAT *instance is satisfiable then there is a labeling to U and V that satisfies all edges of $\mathcal{L}$.*

NO Case: *If the* 3SAT *instance is unsatisfiable then any labeling to U and V satisfies at most $2^{-\gamma_0 r}$ fraction of the edges.*

3 Proof of Theorem 1

In this section we prove the following hardness reduction which implies Theorem 1.

▶ **Theorem 6.** *For any constant integers $Q, k \ge 2$ and $k/2 \ge \ell \ge 1$, and constant $\varepsilon > 0$, there is a polynomial time reduction from* 3SAT *to a Qk-uniform hypergraph G of size n such that:*

YES Case. *If the* 3SAT *instance is satisfiable then there is a k-coloring of the vertices of G such that for in each hyperedge e for some $\ell_e \le \ell$ there exactly ℓ_e colors that appear $(Q-1)$ times each and ℓ_e colors that appear $(Q+1)$ times each, and the other colors appear exactly Q times each. In particular, the hypergraph is k-rainbow colorable.*

NO Case. *If the* 3SAT *instance is not satisfiable then there is no independent set in G of size $1 - \frac{(\ell+1)}{k} + \varepsilon$ fraction of vertices, implying that G is not (k/ℓ)-rainbow colorable.*

In the above, ε can be chosen to be $(\log n)^{-c}$ for some c depending on Q, k and ℓ if $N^{O(\log\log N)}$-time reduction from 3SAT *is allowed.*

The input is an instance $\mathcal{L}$ of LabelCover from Theorem 5 consisting of a bipartite graph $G_{\mathcal{L}}(U,V,E)$, label sets $[M]$ and $[L]$ with $M = dL$, and projections $\{\pi_{vu} : [M] \mapsto [L] \mid \{u,v\} \in E, u \in U, v \in V\}$ such that $|\pi_{vu}^{-1}(j)| = d$ for any $j \in [L]$.

For each vertex $v \in V$, there is a uniformly weighted long code $\mathcal{H}^v$ which is a copy of $[k]^M$. The set of vertices in the output G is $\bigcup_v \mathcal{H}^v$ and the Qk-uniform hyperedges correspond to the PCP test that is described in the next few paragraphs. The parameters Q, k and ℓ in Theorem 6 are fixed in the rest of this section.

3.1 Distributions

First we define $\mathcal{D}(t)$ for $t \le k$ to be the uniform distribution over the set

$$\Gamma(t) := \{z \in [k]^t \mid z_i \ne z_j,\ \forall\ 1 \le i < j \le t\}. \tag{1}$$

Given this, let μ_t be the distribution over $([k]^d)^t$ where $(\overline{x}^{(1)}, \ldots, \overline{x}^{(t)}) \in ([k]^d)^t$ is sampled by independently for each $i \in [d]$ sampling $(\overline{x}_i^{(1)}, \ldots, \overline{x}_i^{(t)})$ from $\mathcal{D}(t)$. Using this we define the distribution $\overline{\mathcal{P}}$ over $\prod_{j=1}^{Q}([k]^d)^k$ by the following sampling procedure.

1. Choose $j^* \in [Q]$ uniformly at random.
2. For each $j \in [Q] \setminus \{j^*\}$ sample $(\overline{x}^{(1,j)}, \ldots, \overline{x}^{(k,j)})$ from μ_k.
3. For j^* sample $(\overline{x}^{(1,j^*)}, \ldots, \overline{x}^{(k,j^*)})$ from $(\mu_\ell \otimes \mu_{k-\ell})$.
4. Ouput $(\overline{x}^{(1,1)}, \ldots, \overline{x}^{(k,1)}, \ldots, \overline{x}^{(1,j)}, \ldots, \overline{x}^{(k,j)}, \ldots, \overline{x}^{(1,Q)}, \ldots, \overline{x}^{(k,Q)})$.

The marginal of $\overline{\mathcal{P}}$ restricted to any $j \in [Q]$ is the same distribution $\mathcal{P}$ where

$$\mathcal{P} := \left(1 - \frac{1}{Q}\right)\mu_k + \left(\frac{1}{Q}\right)(\mu_\ell \otimes \mu_{k-\ell}). \tag{2}$$

With the above in place the PCP test of the verifier is given below.

Test of PCP Verifier. The verifier expects a coloring $C_v : \mathcal{H}^v \to [k]$ for all $v \in V$ and executes the following steps.

1. The verifier chooses a random vertex $u \in U$ and Q of its neighbors $v_1, \ldots, v_Q$ with projections $\pi_j := \pi_{v_j u}$ and long codes $\mathcal{H}^j := \mathcal{H}^{v_j}$. Let the $C_j := C_{v_j}$ be the corresponding colorings.
2. The verifier samples:
$$\left(x^{(1,1)}, \ldots, x^{(k,1)}, \ldots, x^{(1,j)}, \ldots, x^{(k,j)}, \ldots, x^{(1,Q)}, \ldots, x^{(k,Q)}\right) \in \prod_{j=1}^{Q}([k]^{Ld})^k$$
from the distribution $\overline{\mathcal{Q}}$ which is described by sampling independently for each $i \in [L]$,
$$\left(x^{(1,1)}|_{\pi_1^{-1}(i)}, \ldots, x^{(k,1)}|_{\pi_1^{-1}(i)}, \ldots, x^{(1,Q)}|_{\pi_Q^{-1}(i)}, \ldots, x^{(k,Q)}|_{\pi_Q^{-1}(i)}\right)$$
from the distribution $\overline{\mathcal{P}}$ defined above. Let the marginal of $\overline{\mathcal{Q}}$ restricted to $j \in [Q]$ be $\mathcal{Q}_j$ noting that $\mathcal{Q}_j$ is identical to $\mathcal{P}^L$ up to permutation of coordinates.
3. The verifier accepts if the coloring $\left(C_j(x^{(s,j)})\right)_{s\in[k], j\in[Q]}$ has for some $\ell' \leq \ell$, ℓ' colors that appear exactly $(Q-1)$ times each, ℓ' colors that appear exactly $(Q+1)$ times each and the rest of the colors appearing exactly Q times each.

The rest of this section is devoted to proving the YES and NO cases of Theorem 6.

3.2 YES Case

In this case there is a labeling l_v for $v \in V$ such that for any $u \in U$ and its neighbors v, w, $\pi_{vu}(l_u) = \pi_{wu}(l_w)$. Letting $C_v(x) = x_{l_v}$ for $x \in \mathcal{H}^v$ and $v \in V$ yields a coloring of G that makes the verifier accept with probability 1. In particular, this coloring satisfies the YES case of Theorem 6.

3.3 NO Case

Suppose that G contains an independent set $\mathcal{I}$ of $\left(1 - \frac{(\ell+1)}{k} + 4\varepsilon\right)$ fraction of vertices. By standard averaging and using the bi-regularity of the LabelCover instance $\mathcal{L}$ we obtain that for at least ε fraction of "good" vertices $u \in U$, at least ε fraction of its neighbors are "heavy" vertices $v \in V$ which satisfy,

$$\mathbb{E}_{x\in[k]^M}\left[f_v(x)\right] \geq \left(1 - \frac{\ell+1}{k} + \varepsilon\right), \tag{3}$$

where $f_v : [k]^M \to \{0,1\}$ is the indicator of $\mathcal{I} \cap \mathcal{H}^v$.

3.3.1 Lower bound in each Long Code

Fix a choice of a good u and its heavy neighbors $v_1, \ldots, v_Q$ in the verifiers test. For convenience we let $f_j := f_{v_j}$. Fix some $j \in [Q]$, and consider the expectation

$$\mathbb{E}_{(x^{(1,j)},\ldots,x^{(k,j)})\leftarrow\mathcal{Q}_j}\left[\prod_{s=1}^{k} f_j(x^{(s,j)})\right]. \tag{4}$$

By rearranging the coordinates and omitting the subscript j, the above is equivalent to the following expectation:

$$\mathbb{E}_{(x^{(1)},\ldots,x^{(k)})\leftarrow\mathcal{P}^L}\left[\prod_{s=1}^{k} f(x^{(s)})\right], \tag{5}$$

where

$$\mathbb{E}_{x\in([k]^d)^L\simeq[k]^M}\left[f(x)\right] \geq \left(1-\frac{\ell+1}{k}+\varepsilon\right). \tag{6}$$

Using the definition of $\Gamma(.)$ in (1) and by abusing notation slightly let

$$X = (x^1,\ldots,x^\ell) \in (\Gamma(\ell))^M, \quad \text{and} \quad Y = (x^{\ell+1},\ldots,x^k) \in (\Gamma(k-\ell))^M, \tag{7}$$

and define functions

$$A(X) := \prod_{s=1}^{\ell} f(x^s), \quad \text{and} \quad B(Y) := \prod_{s=\ell+1}^{k} f(x^s). \tag{8}$$

Thus, the expectation in (5) is equivalent to

$$\mathbb{E}_{X,Y}\left[A(X)B(Y)\right], \tag{9}$$

where (X,Y) is sampled from $\mathcal{P}^{\otimes L} \simeq [(1-1/Q)\mu_k + (1/Q)(\mu_\ell \otimes \mu_{k-\ell})]^{\otimes L}$. Note that, the marginal distribution of X is $\mu_\ell^{\otimes L}$ and that of Y is $\mu_{k-\ell}^{\otimes L}$. Thus, X and Y are $(1-1/Q)$-correlated. Applying Theorem 2 we obtain,

$$\mathbb{E}_{X,Y}\left[A(X)B(Y)\right] \geq \left(\min\{\mathbb{E}[A(X)],\mathbb{E}[B(Y)]\}\right)^{3Q}. \tag{10}$$

The following argument lower bounds the RHS of the above.

▶ **Lemma 7.** *For A and B defined as above, $\mathbb{E}[A],\ \mathbb{E}[B] \geq \delta_0 := \varepsilon \Big/ \binom{k}{\ell}$.*

Proof. Consider a k-uniform hypergraph H on vertex set $[k]^M$ and hyperedge set $\{(x^1,\ldots,x^k) \mid (x_i^{(1)},\ldots,x_i^{(k)}) \in \Gamma(k),\ \forall i \in [M]\}$. Observe that H is a regular hypergraph i.e. each vertex appears in the same number of hyperedges. Using the bound in (6) along with an averaging, we obtain that at least ε fraction of the hyperedges $(x^{(1)},\ldots,x^{(k)})$ are "dense" satisfying

$$\left|\{s \mid f(x^{(s)}) = 1\}\right| \geq k-\ell. \tag{11}$$

Consider a random choice of $Y = (x^{(\ell+1)},\ldots,x^{(k)})$ sampled from $\mu_{k-\ell}^{\otimes L} \simeq \mathcal{D}(k-\ell)^{\otimes M}$. This is equivalent to a u.a.r choice of a hyperedge in H and a u.a.r subset of $(k-\ell)$ of its vertices. From (11) we obtain $\mathbb{E}_Y[B(Y)] = \mathbb{E}_{(x^{(\ell+1)},\ldots,x^{(k)})\leftarrow\mathcal{D}(k-\ell)^{\otimes M}}\left[\prod_{s=\ell+1}^{k} f(x^{(s)})\right] \geq \delta_0$. Further, since $\ell \leq k/2$ it is easy to see that $\mathbb{E}_X[A(X)] \geq \mathbb{E}_Y[B(Y)]$. ◀

Using Lemma 7 along with (10) we obtain $\mathbb{E}_{X,Y}\left[A(X)B(Y)\right] \geq \delta_0^{3Q}$, which is rewritten as:

$$\mathbb{E}_{(X^{(j)},Y^{(j)})\leftarrow\mathcal{Q}_j}\left[A_j(X^{(j)})B_j(Y^{(j)})\right] \geq \delta_1 := \delta_0^{3Q}. \tag{12}$$

3.3.2 Analyzing over Q Long Codes

Let us fix a choice of u and Q of its neighbors $v_1, \ldots, v_Q$. Using the notation introduced in Section 3.3.1: we have functions $A_j : \Gamma(\ell)^M \to \{0,1\}$ and $B_j : \Gamma(k-\ell)^M \to \{0,1\}$ for $j = 1, \ldots, Q$. From Proposition 3, their Efron-Stein decomposition is given as follows.

$$A_j = \sum_{S \subseteq [M]} A_{j,S}, \quad \text{and} \quad B_j = \sum_{S \subseteq [M]} B_{j,S}. \tag{13}$$

Let R be a parameter to be decided later. Define the following subsets of $(2^{[M]})^Q$:

$$\mathcal{S}_0 := \left\{ (S_1, \ldots, S_Q) \in (2^{[M]})^Q \mid \pi_j(S_j) \cap \pi_{j'}(S_{j'}) = \emptyset,\ 1 \le j < j' \le Q \right\}, \tag{14}$$

$$\mathcal{S}_1 := \left\{ (S_1, \ldots, S_Q) \in (2^{[M]})^Q \setminus \mathcal{S}_0 \mid \max_j |S_j| \le R \right\}, \tag{15}$$

$$\mathcal{S}_2 := \left\{ (S_1, \ldots, S_Q) \in (2^{[M]})^Q \mid \max_j |\pi_j(S_j)| > R^{c_0} \right\}, \tag{16}$$

$$\mathcal{S}_3 := \left\{ (S_1, \ldots, S_Q) \in (2^{[M]})^Q \mid \exists j \text{ s.t. } |S_j| > R, |\pi_j(S_j)| \le R^{c_0} \right\}, \tag{17}$$

where $c_0 > 0$ is the constant from Theorem 5. Note that $\bigcup_{p=0}^{3} \mathcal{S}_p \supseteq (2^{[M]})^Q$. Let us define

$$\delta := \prod_{j=1}^{Q} \mathbb{E}_{(X^{(j)}, Y^{(j)}) \leftarrow \mathcal{Q}_j} \left[A_j(X^{(j)}) B_j(Y^{(j)}) \right]. \tag{18}$$

Since $\mathcal{I}$ is an independent set we also have,

$$\mathbb{E}_{((X^{(1)}, Y^{(1)}), \ldots, (X^{(Q)}, Y^{(Q)})) \leftarrow \overline{\mathcal{Q}}} \left[\prod_{j=1}^{Q} A_j(X^{(j)}) B_j(Y^{(j)}) \right] = 0. \tag{19}$$

Subtracting (19) from (18), expanding the Efron-Stein decomposition and using standard Fourier analysis we obtain

$$\Delta_0 + \Delta_1 + \Delta_2 + \Delta_3 \ge \delta, \tag{20}$$

where,

$$\begin{aligned} \Delta_0 = \sum_{(S_1, \ldots, S_Q) \in \mathcal{S}_0} \Bigg(& \prod_{j=1}^{Q} \mathbb{E}_{(X^{(j)}, Y^{(j)}) \leftarrow \mathcal{Q}_j} \left[A_{j,S_j}(X^{(j)}) B_{j,S_j}(Y^{(j)}) \right] \\ & - \mathbb{E}_{((X^{(1)}, Y^{(1)}), \ldots, (X^{(Q)}, Y^{(Q)})) \leftarrow \overline{\mathcal{Q}}} \left[\prod_{j=1}^{Q} A_{j,S_j}(X^{(j)}) B_{j,S_j}(Y^{(j)}) \right] \Bigg) \end{aligned} \tag{21}$$

and for $p = 1, 2, 3$,

$$\begin{aligned} \Delta_p = \sum_{(S_1, \ldots, S_Q) \in \mathcal{S}_p} \Bigg(& \prod_{j=1}^{Q} \left| \mathbb{E}_{(X^{(j)}, Y^{(j)}) \leftarrow \mathcal{Q}_j} \left[A_{j,S_j}(X^{(j)}) B_{j,S_j}(Y^{(j)}) \right] \right| \\ & + \left| \mathbb{E}_{((X^{(1)}, Y^{(1)}), \ldots, (X^{(Q)}, Y^{(Q)})) \leftarrow \overline{\mathcal{Q}}} \left[\prod_{j=1}^{Q} A_{j,S_j}(X^{(j)}) B_{j,S_j}(Y^{(j)}) \right] \right| \Bigg). \end{aligned} \tag{22}$$

The definition of $\mathcal{S}_0$ and the properties of Efron-Stein decompositions in Proposition 3 imply that each term of the sum in the RHS of (21) is zero. Thus,

$$\Delta_0 = 0. \tag{23}$$

The goal of the rest of the analysis is to show that for an appropriate choice of r in Theorem 5 and the parameter R, the expectation over the choice of u and $v_1, \ldots, v_Q$ of each Δ_p ($p = 1, 2, 3$) is small. Specifically, we shall show that a large $\mathbb{E}[\Delta_1]$ would yield a good labeling to $\mathcal{L}$ contradicting its NO case. Further, Δ_2 is bounded by the dampening induced by the presence of subsets with large projections in its sum, and $\mathbb{E}[\Delta_3]$ is bounded by property (b) of Theorem 5. On the other hand, $\mathbb{E}[\delta]$ is significant due to (12) thereby yielding for us a contradiction in (20).

We begin with the following upper bound on Δ_1.

▶ **Lemma 8.**

$$\Delta_1 \leq 2 \cdot \left(\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}}.$$

Proof. Using $\mathbb{E}[fg] \leq \|f\|_2 \|g\|_2$ observe that

$$\begin{aligned}
\Delta_1 &\leq \sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2 \left\|B_{j,S_j}\right\|_2 + \sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \left\|\prod_{j=1}^{Q} A_{j,S_j}\right\|_2 \left\|\prod_{j=1}^{Q} B_{j,S_j}\right\|_2 \\
&\leq \left(\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}} \left(\sum_{(S_1,\ldots,S_Q)} \prod_{j=1}^{Q} \left\|B_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}} \\
&\quad + \left(\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \left\|\prod_{j=1}^{Q} A_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}} \left(\sum_{(S_1,\ldots,S_Q)} \left\|\prod_{j=1}^{Q} B_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}},
\end{aligned} \tag{24}$$

where the last inequality uses the standard Cauchy-Schwartz inequality. Observe that $\left\|\prod_{j=1}^{Q} A_{j,S_j}\right\|_2^2 = \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2^2$ since $\{X^{(j)}\}_{j=1}^{k}$ are independent.
Similarly, $\left\|\prod_{j=1}^{Q} B_{j,S_j}\right\|_2^2 = \prod_{j=1}^{Q} \left\|B_{j,S_j}\right\|_2^2$. Thus, (24) boils down to,

$$\begin{aligned}
\Delta_1 &\leq 2 \cdot \left(\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}} \left(\sum_{(S_1,\ldots,S_Q)} \prod_{j=1}^{Q} \left\|B_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}} \\
&\leq 2 \cdot \left(\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}} \left(\prod_{j=1}^{Q} \left\|B_j\right\|_2^2 \right)^{\frac{1}{2}} \\
&\leq 2 \cdot \left(\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_1} \prod_{j=1}^{Q} \left\|A_{j,S_j}\right\|_2^2 \right)^{\frac{1}{2}}.
\end{aligned} \tag{25}$$

◀

For analyzing Δ_2 and Δ_3, we use the following lemma which is proved in Appendix A of the full version of this paper [19].

▶ **Lemma 9.**

$$\left|\mathbb{E}_{((X^{(1)},Y^{(1)}),\ldots,(X^{(Q)},Y^{(Q)}))\leftarrow\overline{\mathcal{Q}}}\left[\prod_{j=1}^{Q}A_{j,S_j}(X^{(j)})B_{j,S_j}(Y^{(j)})\right]\right| \leq \left\|\prod_{j=1}^{Q}A_{j,S_j}\right\|_2\left\|\prod_{j=1}^{Q}B_{j,S_j}\right\|_2\left(1-\frac{1}{Q}\right)^{\max_j|\pi_j(S_j)|}.$$

▶ **Corollary 10.**

$$\left|\mathbb{E}_{(X^{(j)},Y^{(j)})\leftarrow\mathcal{Q}_j}\left[A_{j,S_j}(X^{(j)})B_{j,S_j}(Y^{(j)})\right]\right| \leq \left\|A_{j,S_j}\right\|_2\left\|B_{j,S_j}\right\|_2\left(1-\frac{1}{Q}\right)^{|\pi_j(S_j)|}.$$

Proof. Use Lemma 9 with $S_{j'}=\emptyset$ for all $j'\neq j$. ◀

Using the above we have the following bounds for the two sums in Δ_2.

▶ **Claim 11.**

$$\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_2}\left|\mathbb{E}_{\overline{\mathcal{Q}}}\left[\prod_{j=1}^{Q}A_{j,S_j}(X^{(j)})B_{j,S_j}(Y^{(j)})\right]\right| \leq Q\left(1-\frac{1}{Q}\right)^{R^{c_0}}. \tag{26}$$

▶ **Claim 12.**

$$\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_2}\prod_{j=1}^{Q}\left|\mathbb{E}_{\overline{\mathcal{Q}}}\left[A_{j,S_j}(X^{(j)})B_{j,S_j}(Y^{(j)})\right]\right| \leq \left(1-\frac{1}{Q}\right)^{R^{c_0}}. \tag{27}$$

We omit the proofs of Claims 11 and 12 and refer the reader to Appendix B of the full version of this paper [19]. Using Claims 11 and 12 along with $p=2$ in (22) directly yields the following lemma upper bounding Δ_2.

▶ **Lemma 13.**

$$\Delta_2 \leq (Q+1)\left(1-\frac{1}{Q}\right)^{R^{c_0}}.$$

Similarly, we have the following bounds for the two sums in Δ_3.

▶ **Claim 14.**

$$\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_3}\left|\mathbb{E}_{\overline{\mathcal{Q}}}\left[\prod_{j=1}^{Q}A_{j,S_j}(X^{(j)})B_{j,S_j}(Y^{(j)})\right]\right| \leq \sum_{j=1}^{Q}\left(\sum_{\substack{S_j:|S_j|>R\\|\pi_j(S_j)|<R^{c_0}}}\left\|A_{j,S_j}\right\|_2^2\right)^{\frac{1}{2}} \tag{28}$$

▶ **Claim 15.**

$$\sum_{(S_1,\ldots,S_Q)\in\mathcal{S}_3}\prod_{j=1}^{Q}\left|\mathbb{E}_{\overline{\mathcal{Q}}}\left[A_{j,S_j}(X^{(j)})B_{j,S_j}(Y^{(j)})\right]\right| \leq \sum_{j=1}^{Q}\left(\sum_{\substack{S_j:|S_j|>R\\|\pi_j(S_j)|<R^{c_0}}}\left\|A_{j,S_j}\right\|_2^2\right)^{\frac{1}{2}} \tag{29}$$

The proofs of Claims 14 and 15 are given in Appendix C of the full version of this paper [19]. Again, with $p=3$ in (22) Claims 14 and 15 directly imply the following lemma.

▶ **Lemma 16.**

$$\Delta_3 \le 2 \cdot \sum_{j=1}^{Q} \left(\sum_{\substack{S_j : |S_j| > R \\ |\pi_j(S_j)| < R^{c_0}}} \left\| A_{j,S_j} \right\|_2^2 \right)^{\frac{1}{2}} .$$

Plugging in Lemmas 8, 13 and 16 into (20) we obtain that for such a fixed choice of u and $v_1, \ldots, v_Q$

$$\begin{aligned} \left(\frac{\delta}{Q+1} \right) \quad \le \quad & \left(\sum_{(S_1,\ldots,S_Q) \in \mathcal{S}_1} \prod_{j=1}^{Q} \left\| A_{j,S_j} \right\|_2^2 \right)^{\frac{1}{2}} + \sum_{j=1}^{Q} \left(\sum_{\substack{S_j : |S_j| > R \\ |\pi_j(S_j)| < R^{c_0}}} \left\| A_{j,S_j} \right\|_2^2 \right)^{\frac{1}{2}} \\ & + \left(1 - \frac{1}{Q} \right)^{R^{c_0}} . \end{aligned} \tag{30}$$

For a good choice of u, and Q of its heavy neighbors $v_1, \ldots, v_Q$, as defined in (18), $\delta \ge \delta_1^Q$ due to the lower bound in (12). Taking an expectation of (30) over the verifiers choices and noting that with probability at least ε^{Q+1} u is good and $v_1, \ldots, v_Q$ are heavy, we obtain,

$$\begin{aligned} \left(\frac{\delta_1^Q \varepsilon^{Q+1}}{Q+1} \right) \quad \le \quad & \mathbb{E}_{u,\{v_j\}_{j=1}^Q} \left[\left(\sum_{(S_1,\ldots,S_Q) \in \mathcal{S}_1} \prod_{j=1}^{Q} \left\| A_{j,S_j} \right\|_2^2 \right)^{\frac{1}{2}} + \sum_{j=1}^{Q} \left(\sum_{\substack{S_j : |S_j| > R \\ |\pi_j(S_j)| < R^{c_0}}} \left\| A_{j,S_j} \right\|_2^2 \right)^{\frac{1}{2}} \right] \\ & + \left(1 - \frac{1}{Q} \right)^{R^{c_0}} \\ \le \quad & \left(\mathbb{E}_{u,\{v_j\}_{j=1}^Q} \left[\sum_{(S_1,\ldots,S_Q) \in \mathcal{S}_1} \prod_{j=1}^{Q} \left\| A_{j,S_j} \right\|_2^2 \right] \right)^{\frac{1}{2}} \\ & + \sum_{j=1}^{Q} \left(\mathbb{E}_{v_j} \left[\sum_{S_j : |S_j| > R} \left\| A_{j,S_j} \right\|_2^2 \Pr_u \left[|\pi_j(S_j)| < R^{c_0} \right] \right] \right)^{\frac{1}{2}} + \left(1 - \frac{1}{Q} \right)^{R^{c_0}} \\ \le \quad & \left(\mathbb{E}_{u,\{v_j\}_{j=1}^Q} \left[\sum_{(S_1,\ldots,S_Q) \in \mathcal{S}_1} \prod_{j=1}^{Q} \left\| A_{j,S_j} \right\|_2^2 \right] \right)^{\frac{1}{2}} + \frac{Q}{R^{\frac{c_0}{2}}} + \left(1 - \frac{1}{Q} \right)^{R^{c_0}} \end{aligned} \tag{31}$$

where the last inequality uses property (b) of Theorem 5. Consider a labeling to the LabelCover instance $\mathcal{L}$ given by assigning each vertex $v \in V$ label l_v randomly chosen from a subset $S \subseteq [M]$ sampled with probability $\|A_{v,S}\|_2^2$. A vertex $u \in U$ is labeled by uniformly at random choosing $(Q-1)$ of its neighbors $v_2, \ldots, v_Q$, random subsets S_j with probability $\|A_{v_j,S_j}\|_2^2$ independently for $j = 2, \ldots, Q$, and assigning a random label from $\bigcup_{j=2}^{Q} \pi_j(S_j)$. From the definition of $\mathcal{S}_1$ in (15) the expected number of edges satisfied by this strategy is at least

$$\frac{1}{Q} \cdot \frac{1}{R} \cdot \frac{1}{RQ} \cdot \mathbb{E}_{u,\{v_j\}_{j=1}^Q} \left[\sum_{(S_1,\ldots,S_Q) \in \mathcal{S}_1} \prod_{j=1}^{Q} \left\| A_{j,S_j} \right\|_2^2 \right] ,$$

which by (31) is at least

$$\left(\frac{1}{RQ}\right)^2\left[\left(\frac{\delta_1^Q\varepsilon^{Q+1}}{Q+1}\right)-\frac{Q}{R^{\frac{c_0}{2}}}-\left(1-\frac{1}{Q}\right)^{R^{c_0}}\right]^2.$$

For any constant $\varepsilon > 0$, choosing the parameter $R = \text{poly}(1/\varepsilon)$ and $r = \Theta(\log(1/\varepsilon))$ in Theorem 5 yields a contradiction to the NO Case of Theorem 5.

Ruling out $\varepsilon = (\log n)^{-c}$. Choosing $r = (\log\log N)/4$ in Theorem 5 we get that the reduction is of size $n = N^{O(r)}2^{2^{3r}} \leq N^{O(\log\log N)}$. The soundness of $\mathcal{L}$ is $2^{-\Omega(\log\log N)} = 2^{-\Omega(\log\log n)}$. Combining this with the above analysis in the NO Case, choosing $\varepsilon = (\log n)^{-c}$ and $R = \varepsilon^{-c'}$ for some positive constants $c, c' > 0$ (depending on c_0, Q, ℓ, k and γ_0) we obtain a contradiction to the NO Case of Theorem 5.

4 Conclusion

Our work shows that in Qk-uniform k-rainbow colorable hypergraphs ($Q, k \geq 2$) such that in each hyperedge at most 2ℓ of the colors appear $Q \pm 1$ times and the rest exactly Q times, it is NP-hard to find independent sets of density $> (1-(\ell+1)/k)$. It is an open (challenging) question to prove the NP-hardness of finding independent sets of arbitrarily small constant density in such hypergraphs. The question of computing independent sets of density > 0.5 in perfectly balanced rainbow colorable hypergraphs is also similarly open.

References

1. S. Arora, E. Chlamtac, and M. Charikar. New approximation guarantee for chromatic number. In *Proc. STOC*, pages 215–224, 2006.
2. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.
3. S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.
4. P. Austrin, V. Guruswami, and J. Håstad. $(2+\varepsilon)$-sat is NP-hard. In *Proc. FOCS*, pages 1–10, 2014.
5. N. Bansal. Constructive algorithms for discrepancy minimization. In *Proc. FOCS*, pages 3–10, 2010.
6. A. Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41(3):470–516, 1994.
7. A. Blum and D. R. Karger. An $\tilde{O}(n^{3/14})$-coloring algorithm for 3-colorable graphs. *Information Processing Letters*, 61(1):49–53, 1997.
8. B. Bollobás, D. Pritchard, T. Rothvoß, and A. D. Scott. Cover-decomposition and polychromatic numbers. *SIAM Journal on Discrete Mathematics*, 27(1):240–256, 2013.
9. J. Brakensiek and V. Guruswami. The quest for strong inapproximability results with perfect completeness. In *Proc. APPROX-RANDOM*, pages 4:1–4:20, 2017. URL: `https://eccc.weizmann.ac.il/report/2017/080`.
10. M. Charikar, A. Newman, and A. Nikolov. Tight hardness results for minimizing discrepancy. In *Proc. SODA*, pages 1607–1614, 2011.
11. H. Chen and A. M. Frieze. Coloring bipartite hypergraphs. In *Proc. IPCO*, pages 345–358, 1996.
12. I. Dinur and V. Guruswami. PCPs via low-degree long code and hardness for constrained hypergraph coloring. In *Proc. FOCS*, 2013.

13 I. Dinur, O. Regev, and C. D. Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005.

14 V. Guruswami, J. Håstad, P. Harsha, S. Srinivasan, and G. Varma. Super-polylogarithmic hypergraph coloring hardness via low-degree long codes. *SIAM Journal of Computing*, 46(1):132–159, 2017.

15 V. Guruswami, J. Håstad, and M. Sudan. Hardness of approximate hypergraph coloring. *SIAM Journal of Computing*, 31(6):1663–1686, 2002.

16 V Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM Journal of Discrete Mathematics*, 18(1):30–40, 2004.

17 V. Guruswami and E. Lee. Strong inapproximability results on balanced rainbow-colorable hypergraphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:43, 2014. URL: `http://eccc.hpi-web.de/report/2014/043`.

18 V. Guruswami and E. Lee. Strong inapproximability results on balanced rainbow-colorable hypergraphs. In *Proc. SODA*, pages 822–836, 2015.

19 V. Guruswami and R. Saket. Hardness of rainbow coloring hypergraphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 2017. URL: `https://eccc.weizmann.ac.il/report/2017/147/`.

20 E. Halperin, R. Nathaniel, and U. Zwick. Coloring k-colorable graphs using relatively small palettes. *Journal of Algorithms*, 45(1):72–90, 2002.

21 J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

22 J. Holmerin. Vertex cover on 4-regular hyper-graphs is hard to approximate within 2 - ε. In *Proc. CCC*, 2002.

23 S. Huang. $2^{(\log n)^{1/10-o(1)}}$ hardness for hypergraph coloring. *CoRR*, abs/1504.03923, 2015.

24 D. R. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998.

25 K. Kawarabayashi and M. Thorup. Combinatorial coloring of 3-colorable graphs. In *Proc. FOCS*, pages 68–75, 2012.

26 P. Kelsen, S. Mahajan, and R. Hariharan. Approximate hypergraph coloring. In *Proc. SWAT*, pages 41–52, 1996.

27 S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.

28 S. Khot. Hardness results for coloring 3-colorable 3-uniform hypergraphs. In *Proc. FOCS*, pages 23–32, 2002.

29 S. Khot. On the power of unique 2-prover 1-round games. In *Proc. STOC*, pages 767–775, 2002.

30 S. Khot and R. Saket. Hardness of finding independent sets in 2-colorable and almost 2-colorable hypergraphs. In *Proc. SODA*, pages 1607–1625, 2014.

31 S. Khot and R. Saket. Hardness of coloring 2-colorable 12-uniform hypergraphs with $2^{(\log n)^{\Omega(1)}}$ colors. *SIAM Journal of Computing*, 46(1):235–271, 2017.

32 M. Krivelevich, R. Nathaniel, and B. Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. *Journal of Algorithms*, 41(1):99–113, 2001.

33 S. Lovett and R. Meka. Constructive discrepancy minimization by walking on the edges. *SIAM Journal of Computing*, 44(5):1573–1582, 2015.

34 E. Mossel. Gaussian bounds for noise correlation of functions. *Geometric and Functional Analysis*, 19:1713–1756, 2010.

35 E. Mossel, K. Oleszkiewicz, and A. Sen. On reverse hypercontractivity. *Geometric and Functional Analysis*, 23(3):1062–1097, 2013.

36 R. Raz. A parallel repetition theorem. *SIAM Journal of Computing*, 27(3):763–803, 1998.

37 R. Saket. Hardness of finding independent sets in 2-colorable hypergraphs and of satisfiable CSPs. In *Proc. CCC*, pages 78–89, 2014.

38 G. Varma. Reducing uniformity in Khot-Saket hypergraph coloring hardness reductions. *Chicago J. Theor. Comput. Sci.*, 2016, 2016.

39 A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, 1983.

Network Construction with Ordered Constraints

Yi Huang[1], Mano Vikash Janardhanan[2], and Lev Reyzin*[3]

1 **Department of Mathematics, Statistics, and Computer Science**
University of Illinois at Chicago, Chicago IL, USA
yhuang89@uic.edu
2 **Department of Mathematics, Statistics, and Computer Science**
University of Illinois at Chicago, Chicago IL, USA
mjanar2@uic.edu
3 **Department of Mathematics, Statistics, and Computer Science**
University of Illinois at Chicago, Chicago IL, USA
lreyzin@uic.edu

Abstract

In this paper, we study the problem of constructing a network by observing ordered connectivity constraints, which we define herein. These ordered constraints are made to capture realistic properties of real-world problems that are not reflected in previous, more general models. We give hardness of approximation results and nearly-matching upper bounds for the offline problem, and we study the online problem in both general graphs and restricted sub-classes. In the online problem, for general graphs, we give exponentially better upper bounds than exist for algorithms for general connectivity problems. For the restricted classes of stars and paths we are able to find algorithms with optimal competitive ratios, the latter of which involve analysis using a potential function defined over PQ-trees.

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

Keywords and phrases subgraph connectivity, network construction, ordered connectivity constraints, PQ-trees

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.34

1 Introduction

In this paper, we study the problem of recovering a network after observing how information propagates through the network. Consider how a tweet (through "retweeting" or via other means) propagates through the Twitter network – we can observe the identities of the people who have retweeted it and the timestamps when they did so, but may not know, for a fixed user, via whom he got the original tweet. So we see a chain of users for a given tweet. This chain is semi-ordered in the sense that, each user retweets from some one before him in the chain, but not necessarily the one directly before him. Similarly, when a virus such as Ebola spreads, each new patient in an outbreak is infected from some one who has previously been infected, but it is often not immediately clear from whom.

In a graphical social network model with nodes representing users and edges representing links, an "outbreak" illustrated above is captured exactly by the concept of an **ordered constraint** which we will define formally below. One could hope to be able to learn something about the structure of the network by observing repeated outbreaks, or a sequence of ordered constraints.

* This research was supported in part by ARO grant 66497-NS.

Formally we call our problem **Network Construction with Ordered Constraints** and define it as follows. Let $V = \{v_1, \ldots, v_n\}$ be a set of vertices. An **ordered constraint** $\mathcal{O}$ is an ordering on a subset of V of size $s \geq 2$. The constraint $\mathcal{O} = (v_{k_1}, \ldots, v_{k_s})$ is satisfied if for any $2 \leq i \leq s$, there exists at least one $1 \leq j < i$ such that the edge $e = \{v_{k_j}, v_{k_i}\}$ is included in a solution. Given a collection of ordered constraints $\{\mathcal{O}_1, \ldots, \mathcal{O}_r\}$, the task is to construct a set E of edges among the vertices V such that all the ordered constraints are satisfied and $|E|$ is minimized.

We can see that our newly defined problem resides in a middle ground between path constraints, which are too rigid to be very interesting, and the well-studied subgraph connectivity constraints [8, 18, 19], which are more relaxed. The established subgraph connectivity constraints problem involves getting an arbitrary collection of connectivity constraints $\{S_1, \ldots, S_r\}$ where each $S_i \subset V$ and requires vertices in a given constraint to form a connected induced subgraph; we will occasionally refer to these as **unordered** or **general constraints**. The task is to construct a set E of edges satisfying the connectivity constraints such that $|E|$ is minimized.

We want to point out one key observation relating the ordered constraint to the connectivity constraint – an ordered constraint $\mathcal{O} = (v_{k_1}, \ldots, v_{k_s})$ is equivalent to $s - 1$ connectivity constraints $S_2, \ldots, S_s$, where $S_i = \{v_{k_1}, \ldots, v_{k_i}\}$. We note that this observation plays an important role in several proofs in this paper which employ previous results on subgraph connectivity constraints – in particular, upper bounds from the more general case can be used in the ordered case (with some overhead), and our lower bounds apply to the general problem.

In the offline version of the Network Construction with Ordered Constraints problem, the algorithm is given all of the constraints all at once; in the **online** version of the problem, the constraints are given one by one to the algorithm, and edges must be added to satisfy each new constraint when it is given. Edges cannot be removed.

An algorithm is said to be **c-competitive** if the cost of its solution is less than c times OPT, where OPT is the best solution in hindsight (c is also called the competitive ratio). When we restrict the underlying graph in a problem to be a class of graphs, e.g. trees, we mean all the constraints can be satisfied, in an optimal solution (for the online case, in hindsight), by a graph from that class.

1.1 Past Work

In this paper we study the problem of network construction from ordered constraints. This is an extension of the more general model where constraints come unordered.

For the general problem, Korach and Stern [18] had some of the initial results, in particular for the case where the constraints can be optimally satisfied by a tree, they give a polynomial time algorithm that finds the optimal solution. In subsequent work, in [19] Korach and Stern considered this problem for the even more restricted problem where the optimal solution forms a tree, and all of the connectivity constraints must be satisfied by stars.

Then, Angluin et al. [8] studied the general problem, where there is no restriction on structure of the optimal solution, in both the offline and online settings. In the offline case, they gave nearly matching upper and lower bounds on the hardness of approximation for the problem. In the online case, they give a $O(n^{2/3} \log^{2/3} n)$-competitive algorithm against oblivious adversaries; we show that this bound can be drastically improved in the ordered version of the problem. They also characterized special classes of graphs, i.e. stars and paths, which we are also able to do herein for the ordered constraint case. Independently of that work, Chockler et al. [13] also nearly characterized the offline general case.

In a different line of work Alon et al. [3] explore a wide range of network optimization problems; one problem they study involves ensuring that a network with fractional edge weights has a flow of 1 over cuts specified by the constraints. Alon et al. [2] also study approximation algorithms for the Online Set Cover problem which have been shown by Angluin et al. [8] to have connections with Network Construction problems.

In related areas, Gupta et al. [17] considered a network design problem for pairwise vertex connectivity constraints. Moulin and Laigret [20] studied network connectivity constraints from an economics perspective. Another motivation for studying this problem is to discover social networks from observations. This and similar problems have also been studied in the learning context [7, 5, 15, 24].

Finally, in query learning, the problem of discovering networks from connectivity queries has been much studied [1, 4, 9, 10, 16, 23]. In active learning of hidden networks, the object of the algorithm is to learn the network exactly. Our model is similar, except the algorithm only has the constraints it is given, and the task is to output the cheapest network consistent with the constraints.

1.2 Connection to network inference

This model is also known to have connections to network inference [6, 22]. Let $p_{(u,v)}$ be the a priori probability of an edge appearing between nodes u and v. If $p_{u,v}$'s are $\leq 1/2$ and are independent, the maximum likelihood social network given the constraints is a set of edges E that satisfies all of the constraints and maximises the following quantity

$$\prod_{\{u,v\}\in E} p_{(u,v)} \prod_{\{u,v\}\notin E} \left(1-p_{(u,v)}\right) = \prod_{\{u,v\}} \left(1-p_{(u,v)}\right) \prod_{\{u,v\}\in E} \frac{p_{(u,v)}}{\left(1-p_{(u,v)}\right)}$$

Taking the logarithm, we want a set of edges E that minimizes the sum

$$\sum_{\{v,u\}\in E} -\log\left(\frac{p_{(u,v)}}{\left(1-p_{(u,v)}\right)}\right).$$

The assumption that for all u, v, $p_{(u,v)} \leq 1/2$ implies that each term, or cost, in the sum is non-negative.

1.3 Our results

In Section 2, we examine the offline problem, and show that the Network Construction problem is NP-Hard to approximate within a factor of $\Omega(\log n)$. A nearly matching upper bound comes from Theorem 2 of Angluin et al. [8].

In Section 3, we study online problem. For problems on n nodes, for r constraints, we give an $O\left((\log r + \log n)\log n\right)$ competitive algorithm against oblivious adversaries, and an $\Omega(\log n)$ lower bound (Section 3.1).

Then, for the special cases of stars and paths (Sections 3.2 and 3.3), we find asymptotic optimal competitive ratios of 3/2 and 2, respectively. The proof of the latter uses a detailed analysis involving PQ-trees [11]. The competitive ratios are asymptotic in n.

2 The offline problem

In this section, we examine the Network Construction with Ordered Constraints problem in the offline case. We are able to obtain the same lower bound as Angluin et al. [8] in the general connectivity constraints case.

▶ **Theorem 1.** *If P≠NP, the approximation ratio of the* ***Network Construction with Ordered Constraints*** *problem is* $\Omega(\log n)$.

Proof. We prove the theorem by reducing from the Hitting Set problem. Let $(U, \mathcal{S})$ be a hitting set instance, where $U = \{u_1, \ldots, u_n\}$ is the universe, and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a set of subsets of U. A subset $H \subset U$ is called a hitting set if $H \cap S_i \neq \emptyset$ for $i = 1, \ldots, m$. The objective of the Hitting Set problem is to minimize $|H|$. We know from [14, 21] that the Hitting Set problem cannot be approximated by any polynomial time algorithm within a ratio of $o(\log n)$ unless P=NP. Here we show that the Network Construction problem is inapproximable better than an $O(\log n)$ factor by first showing that we can construct a corresponding Network Construction instance to any given Hitting Set instance, and then showing that if there is a polynomial time algorithm that can achieve an approximation ratio $o(\log n)$ to the Network Construction problem, then the Hitting Set problem can also be approximated within in a ratio of $o(\log n)$, which is a contradiction.

We first define a Network Construction instance, corresponding to a given Hitting Set instance $(U, \mathcal{S})$, with vertex set $U \cup W$, where $W = \{w_1, \ldots, w_{n^c}\}$ for some $c > 2$. Note that we use the elements of the universe of hitting set instance as a part of the vertex set of Network Construction instance. The ordered constraints are the union of the following two sets:

- $\{(u_i, u_j)\}_{1 \leq i < j \leq n}$;
- $\{(S_k, w_l)\}_{S_k \in \mathcal{S}, 1 \leq l \leq n^c}$,

where by (S_k, w_l) we mean an ordered constraint with all vertices except the last one from a subset S_k of U, while the last vertex w_l is an element in W. The vertices from S_k are ordered arbitrarily.

We note that the first set of ordered constraints forces a complete graph on U, and the second set of ordering demands that there is at least one edge going out from each S_k connecting each element in W. Let $\mathcal{A}$ be an algorithm solving the Network Construction problem, and let E_l denote the set of edges added by $\mathcal{A}$ incident to w_l. Because of the second set of ordered constraints, the set $H_l = \{u \in U | \{u, w_l\} \in E_l\}$ is a hitting set of $\mathcal{S}$!

Let $H \subset U$ be any optimal solution to the hitting set instance, and denote by $\mathrm{OPT_H}$ the size of H. It is easy to see the two sets of ordered constraints can be satisfied by putting a complete graph on U and a complete bipartite graph between H and W. Hence the optimal solution to the Network Construction instance satisfies

$$\mathrm{OPT} \leq \binom{n}{2} + n^c \, \mathrm{OPT_H},$$

where OPT is the minimum number of edges needed to solve the Network Construction instance. Let us assume that there is a polynomial time approximation algorithm to the Network Construction problem that adds ALG edges. Without loss of generality we can assume that the algorithm adds no edge among vertices in W, because any edge within W can be removed without affecting the correctness of the solution, which implies that $\mathrm{ALG} = \binom{n}{2} + \sum_{l=1}^{n^c} |E_l|$. Now if ALG is $o(\log n \cdot \mathrm{OPT})$, from the fact that $|H_l| = |E_l|$, we get

$$\begin{aligned} \min_{1 \leq l \leq n^c} |H_l| \leq \frac{\mathrm{ALG} - \binom{n}{2}}{n^c} &= \frac{o\left(\log n \left(\binom{n}{2} + n^c \, \mathrm{OPT_H}\right)\right) - \binom{n}{2}}{n^c} \\ &= o\left(\log n \cdot \mathrm{OPT_H}\right), \end{aligned}$$

which means by finding the smallest set H_{l_0} among all the H_ls, we get a hitting set that has size within an $o(\log n)$ factor of the optimal solution to the Hitting Set instance, which is a contradiction. ◀

We also observe that the upper bound from the more general problem implies a bound in our ordered case. We note the upper and lower bounds match when $r = poly(n)$.

▶ **Corollary 2** (of Theorem 2 from Angluin et al. [8]). *There is a polynomial time $O(\log r + \log n)$-approximation algorithm for the Network Construction with Ordered Constraints problem on n nodes and r ordered constraints.*

Proof. Observing that r ordered constraints imply at most nr unordered constraints on a graph with n nodes, we can use the $O(\log nr)$ upper bound from Angluin et al. [8]. ◀

3 The online problem

Here, we study the online problem, where constraints come in one at a time, and the algorithm must satisfy them by adding edges as the constraints arrive.

3.1 Arbitrary graphs

▶ **Theorem 3.** *There is an $O((\log r + \log n) \log n)$ upper bound for the competitive ratio for the **Online Network Construction with Ordered Constraints** problem on n nodes and r ordered constraints against an oblivious adversary.*

Proof. To prove the statement, we first define the **Fractional Network Construction** problem, which has been shown by Angluin et al. [8] to have an $O(\log n)$-approximation algorithm. The upper bound is then obtained by applying a probabilistic rounding scheme to the fractional solution given by the approximation. The proof heavily relies on arguments developed by Buchbinder and Naor [12], and Angluin et al. [8].

In the **Fractional Network Construction** problem, we are also given a set of vertices and a set of constraints $\{S_1, \ldots, S_r\}$ where each S_i is a subset of the vertex set. Our task is to assign weights w_e to each edge e so that the maximum flow between each pair of vertices in S_i is at least 1. The optimization problem is to minimize $\sum w_e$. Since subgraph connectivity constraint is equivalent to requiring a maximum flow of 1 between each pair of vertices with edge weight $w_e \in \{0, 1\}$, the fractional network construction problem is the linear relaxation of the subgraph connectivity problem. Lemma 2 of Angluin et al. [8] gives an algorithm that multiplicatively updates the edge weights until all the flow constraints are satisfied. It also shows that the sum of weights given by the algorithm is upper bounded by $O(\log n)$ times the optimum.

As we pointed out in the introduction, an ordered constraint $\mathcal{O}$ is equivalent to a sequence of subgraph connectivity constraints. So in the first step, we feed the r sequences of connectivity constraints, each one is equivalent to an ordered constraint, to the approximation algorithm to the fractional network construction problem and get the edge weights. Then we apply a rounding scheme similar to the one considered by Buchbinder and Naor [12] to the weights. For each edge e, we choose t random variables $X(e, i)$ independently and uniformly from $[0, 1]$, and let the threshold $T(e) = \min_{i=1}^{t} X(e, i)$. We add e to the graph if $w_e \geq T(e)$.

Since the rounding scheme has no guarantee to produce a feasible solution, the first thing we need to do is to determine how large t should be to make all the ordered constraints satisfied with high probability.

We note that an ordered constraint $\mathcal{O}_i = \{v_{i1}, v_{i2}, \ldots, v_{is_i}\}$ is satisfied if and only if the $(s_i - 1)$ connectivity constraints $\{v_{i1}, v_{i2}\}, \ldots, \{v_{i1}, \ldots, v_{is_i-1}, v_{is_i}\}$ are satisfied, which is equivalent, in turn, to the fact that there is an edge that goes across the $(\{v_{i1}, \ldots, v_{ij-1}\}, \{v_{ij}\})$ cut, for $2 \leq j \leq s_i$. For any fixed cut C, the probability the cut is not crossed equals

$\prod_{e\in C}(1-w_e)^t \le \exp\left(-t\sum_{e\in C} w_e\right)$. By the max-flow min-cut correspondence, we know that $\sum_{c\in C} w_e \ge 1$ in the fractional solution given by the approximation algorithm for all cuts $C = (\{v_{i1},\dots,v_{ij-1}\},\{v_{ij}\})$, $1 \le i \le r$, $2 \le j \le s_i$, and hence the probability that there exists at least one unsatisfied $\mathcal{O}_i$ is upper bounded by $rn\exp(-t)$. So $t = c(\log n + \log r)$, for any $c > 1$, makes the probability that the rounding scheme fails to produce a feasible solution approaches 0 as n increases.

Because the probability that e is added equals the probability that at least one $X(e,i)$ is less than w_e, and hence is upper bounded by $w_e t$, we get the expected number of edges added is upper bounded by $t\sum w_e$ by linearity of expectation. Since the fractional solution is upper bounded by $O(\log n)$ times the optimum of the fractional problem, which is upper bounded by any integral solution, our rounding scheme gives a solution that is $O((\log r + \log n)\log n)$ times the optimum. ◀

▶ **Corollary 4.** *If the number of ordered constraints $r = \mathrm{poly}(n)$, then the algorithm above gives an $O\left((\log n)^2\right)$ upper bound for the competitive ratio against an oblivious adversary.*

▶ Remark. We can generalize Theorem 3 to the weighted version of the Online Network Construction with Ordered Constraints problem. In the weighted version, each edge $e = (u,v)$ is associated with a cost c_e and the task is to select edges such that the connectivity constraints are satisfied and $\sum c_e w_e$ is minimised where $w_e \in \{0,1\}$ is a variable indicating whether an edge is picked or not and c_e is the cost of the edge. The same approach in the proof of Theorem 3 gives an upper bound of $O((\log r + \log n)\log n)$ for the competitive ratio of the weighted version of the Online Network Construction with Ordered Constraints problem.

For an lower bound for the competitive ratio for the Online Network Construction with Ordered Constraints problem against an oblivious adversary, we study the **Online Permutation Hitting Set** problem defined below: Let k be a positive integer, and let π be a permutation on $[k]$. Define sets

$$P_\pi^i = \{\pi(1),\pi(2),\dots,\pi(i)\} \quad \text{for } i = 1,\dots,k,$$

▶ **Definition 5 (Online Permutation Hitting Set).** Let π be a permutation on $[k]$ and let $\left(P_\pi^k, P_\pi^{k-1},\dots,P_\pi^1\right)$ arrive one at a time. A solution to the Online Permutation Hitting Set problem is a sequence of set $H_1 \subset H_2 \subset \cdots \subset H_k$ such that $H_i \cap P_\pi^{k+1-i} \neq \emptyset$ for $i = 1,\dots,k$.

▶ **Lemma 6.** *The expected size of H_k for any algorithm that solves the **Online Permutation Hitting Set** problem over the space of all permutations on $[k]$ under the uniform distribution is lower bounded by $h_k = \sum_{i=1}^k \frac{1}{i}$.*

Proof. We first show that the lower bound is achieved by a randomized algorithm $\mathcal{A}^0$, and then we show that no other randomized algorithm could do better than $\mathcal{A}^0$.

We start by describing $\mathcal{A}^0$. Upon seeing P_π^{k+1-i}, $\mathcal{A}^0$ sets H_i using the following rule:

$$H_i = \begin{cases} H_{i-1} & \text{if } H_{i-1}\cap P_\pi^{k+1-i} \neq \emptyset \\ H_{i-1}\cup\{a\} & \text{if } H_{i-1}\cap P_\pi^{k+1-i} = \emptyset \end{cases},$$

where a is a random point in P_π^{k+1-i}. Let E_i denote the expected size of the final output of $\mathcal{A}^0$ on permutations on $[i]$ for all $i = 1,\dots,k$. By symmetry, without loss of generality, we assume that $\mathcal{A}^0$ add 1 upon receiving $P_\pi^k = [k]$, then we have

$$E_k = 1 + \sum_{i=1}^k E_i \mathbb{P}(\pi(i) = 1) = 1 + \frac{1}{k}\sum_{i=1}^{k-1} E_i$$

The first equality is because $\mathcal{A}^0$ doesn't need to add more points until it receives P_π^{k-i} if $\pi(i) = 1$, and the second equality is because that all i has the same probability to be mapped to 1. To show that $E_k = h_k$, we first verify that the harmonic series satisfies the same recursive relation. In fact, we have

$$\begin{aligned} 1 + \frac{1}{k}\sum_{i=1}^{k-1} h_i &= \frac{1}{k}\left(k + \sum_{i=1}^{k-1}\sum_{j=1}^{i}\frac{1}{j}\right) = \frac{1}{k}\left(k + \sum_{i=1}^{k-1}(k-i)\cdot\frac{1}{i}\right) \\ &= \frac{1}{k}\left(k + \sum_{i=1}^{k-1}\left(k\cdot\frac{1}{i} - 1\right)\right) = \frac{1}{k}\left(1 + k\sum_{i=1}^{k-1}\frac{1}{i}\right) = h_k. \end{aligned}$$

Since $E_1 = h_1 = 1$, we have $E_k = h_k$ for all $k \in \mathbb{N}^+$.

Next, we show that $\mathcal{A}^0$ is in fact the best randomized algorithm in expectation. To this end, we need to show two things:

(i) when $H_{i-1} \cap P_\pi^{k+1-i} \neq \emptyset$, adding point(s) is counter-productive;
(ii) when $H_{i-1} \cap P_\pi^{k+1-i} = \emptyset$, adding more than one points is counter-productive.

To show i., let us assume that $H_{i-1} \cap P_\pi^{k+1-i} \neq \emptyset$ and j is the smallest integer in $1 \leq j \leq k+1-i$ such that $\pi(j)$ is contained in H_{i-1}. We show that adding one more point hurts the expectation, and the proof for adding more than one points follows the same fashion. $\mathcal{A}^0$ will wait till P_π^{j-1} and add a random point from $\{\pi(1), \pi(2), \ldots, \pi(j-1)\}$. Hence, $\mathcal{A}^0$ does not assign probabilities to any point in $\{\pi(j+1), \ldots, \pi(k+1-i)\}$. We note that adding any point in $\{\pi(j+1), \ldots, \pi(k+1-i)\}$ wouldn't help. Hence, any algorithm that assigns non-zero probabilities to $\{\pi(j+1), \ldots, \pi(k+1-i)\}$ will do worse than $\mathcal{A}^0$ in expectation.

To show ii., for simplicity, we show that adding two points upon seeing P_π^k hurts the expectation, and the proof for adding more than two points follows the same fashion. We show this by induction assuming that $\mathcal{A}^0$ is the best algorithm in expectation for all $i < k$. By symmetry, without loss of generality, we assume that $H_1 = \{1, 2\}$. We have

$$\begin{aligned} & \mathbb{E}\left(|H_k| | H_1 = \{1,2\}\right) \\ = \; & 2 + \left(\sum \mathbb{E}\left(|H_k| | \pi^{-1}(1) < \pi^{-1}(2)\right)\right)\mathbb{P}\left(\pi^{-1}(1) < \pi^{-1}(2)\right) \\ & + \left(\sum \mathbb{E}\left(|H_k| | \pi^{-1}(1) > \pi^{-1}(2)\right)\right)\mathbb{P}\left(\pi^{-1}(1) > \pi^{-1}(2)\right) \\ \geq \; & 2 + 2\cdot\frac{1}{2}\left(\frac{1}{k-1}\sum_{i=1}^{k-2} E_i\right) = 2 + \frac{1}{k-1}\sum_{i=1}^{k-2} E_i = 1 + E_{k-1} > E_k, \end{aligned}$$

where the first inequality follows from i. and the inductive hypothesis. ◀

With the help of Lemma 6, we can prove the following lower bound.

▶ **Theorem 7.** *There exists an $\Omega(\log n)$ lower bound for the competitive ratio for the **Online Network Construction with Ordered Constraints** problem against an oblivious adversary.*

Proof. The adversary divides the vertex set into two parts U and V, where $|U| = \sqrt{n}$ and $|V| = n - \sqrt{n}$, and gives the constraints as follows. First, it forces a complete graph in U by giving the constraint $\{u_i, u_j\}$ for each pair of vertices $u_i, u_j \in U$. At this stage both the algorithm and optimal solution will have a clique in U, which costs $\Theta(n)$.

Then, for each $v \in V$, first fix a random permutation π_v on U and give the ordered constraints

$$\mathcal{O}_{(v,i)} = (\pi_v(1), \pi_v(2), \ldots, \pi_v(i), v) \quad \text{for } i = 1, \ldots, \sqrt{n}.$$

First note that all these constraints can be satisfied by adding $e_v = \{\pi_v(1), v\}$ for each $v \in V$ which costs $\Theta(n)$. However, the adversary gives constraints in the following order:

$$\mathcal{O}_{(v,\sqrt{n})}, \mathcal{O}_{(v,\sqrt{n}-1)}, \ldots, \mathcal{O}_{(v,1)}.$$

To satisfy $O_{(v,i)}$, any algorithm just need to make sure that at least one point in P_π^i is chosen to connect to v, and hence the algorithm is in fact solving an instance of the Online Permutation Hitting Set problem on input π at this stage. By Proposition 6, we know that all algorithm solving the Online Permutation Hitting Set problem will add $\Omega(\log \sqrt{n})$ points in expectation, and this shows that any algorithm to the Online Network Construction with Ordered Constraints problem needs to add $\Omega(n + n \log n)$ edges in expectation in total. This gives us the desired result because $\mathrm{OPT} = O(n)$. ◀

Now we study the online problem when it is known that an optimal graph can be a star or a path. These special cases are challenging in their own right and are often studied in the literature to develop more general techniques [8].

3.2 Stars

▶ **Theorem 8.** *The optimal competitive ratio for the* ***Online Network Construction with Ordered Constraints*** *problem when the algorithm knows that the optimal solution forms a* ***star*** *is asymptotically* $3/2$.

Proof. For the lower bound, we note that the adversary can simply give $\mathcal{O}_i = (v_1, v_2, v_i)$ for all $i = 3, \ldots, n$ obliviously for the first $n - 2$ rounds. Then an algorithm, besides adding $\{v_1, v_2\}$ in the first round, can only choose from adding either $\{v_1, v_i\}$ or $\{v_2, v_i\}$, or both in each round. Note that v_1 or v_2 have to be the center since the first constraint ensures that $\{v_1, v_2\}$ is an edge. After the first $n - 2$ rounds, the adversary counts the number of v_1 and v_2's neighbors, and chooses the one with fewer neighbors, say v_1, to be the center by giving the constraints (v_1, v_i) for all $i = 3, \ldots, n$ where no edge (v_1, v_i) exists. Since the algorithm has to add at least $\lceil (n-2)/2 \rceil$ edges that are unnecessary in the hindsight, we get an asymptotic lower bound $3/2$.

For the upper bound, assume that either v_1 or v_2 is the center and the first ordered constraint is $\mathcal{O}_1$ is $(v_1, v_2, \ldots)$, the algorithm works as follows:

1. It adds $\{v_1, v_2\}$ in the first round.
2. For any constraint (including the first) that starts with v_1 and v_2, the algorithm always adds edges of the form (v_1, v_k) or (v_2, v_k) where $k \neq 1, 2$ in such a way that the following two conditions hold:
   - Degree of v_1 and degree of v_2 differ by at most 1.
   - The degree of v_k is 1 for $k \neq 1, 2$
3. Upon seeing a constraint that does not start with v_1 and v_2, which reveals the center of the star, it connects the center to all vertices that are not yet connected to the center.

Since the algorithm adds, at most $n/2 - 1$ edges to the wrong center, this gives us an asymptotic upper bound $3/2$, which matches the lower bound. ◀

Algorithm 1 Forcing pre-degree to be at least 2

Give ordered constraint $\mathcal{O} = (v_1, v_2, v_3, \ldots, v_n)$ to the algorithm;
for $i = 3$ to n **do**
  if the pre-degree of v_i is at least 2 **then**
    continue;
  else
    pick a path on $\{v_1, v_2, v_3, \ldots, v_{i-1}\}$ (say P_i) that satisfies all the constraints up to this round (the existence of P_i follows by induction) and an endpoint u of the path that is not connected to v_i, and give the algorithm the constraint (v_i, u);
  end if
end for

3.3 Paths

In the next two theorems, we give matching lower and upper bounds (in the limit) for path graphs.

▶ **Theorem 9.** *The **Online Network Construction with Ordered Constraints** problem when the algorithm knows that the optimal solution forms a **path** has an asymptotic lower bound of* 2 *for the competitive ratio.*

Proof. Let us name the vertices $v_1, v_2, v_3, \ldots, v_n$. For $3 \leq i \leq n$, define the **pre-degree** of a vertex v_i to be the number of neighbors v_i has in $\{v_1, v_2, v_3, \ldots, v_{i-1}\}$. Algorithm 1 below is a simple strategy the adversary can take to force $v_3, \ldots, v_n$ to all have pre-degree at least 2. Since any algorithm will add at least $2n - 3$ edges, this gives an asymptotic lower bound of 2.

Suppose P_i was the path picked in round i (i.e. P_i satisfies all constraints up to round i). Then, P_i along with the edge (v_i, u) is a path that satisfies all constraints up to round $i + 1$. Hence by induction, for all i, there is a path that satisfies all constraints given by the adversary up to round i. ◀

▶ **Theorem 10.** *The competitive ratio for the **Online Network Construction with Ordered Constraints** problem when the algorithm knows that the optimal solution forms a **path** has an asymptotic upper bound of* 2.

Proof. For our algorithm matching the upper bound, we use the PQ-trees, introduced by Booth and Lueker [11], which keep track all consistent permutations of vertices given contiguous intervals of vertices, since vertices in any ordered constraint form a contiguous interval if the underlying graph is a path. Our analysis is based on ideas from Angluin et al. [8], who also use PQ-trees for analyzing the general problem.[1]

A **PQ-tree** is a tree whose leaf nodes are the vertices and each internal node is either a **p-node** or a **q-node**.

- A **p-node** has two or more children of any type. The children of a p-node form a contiguous interval that can be in any order.
- A **q-node** has three or more children of any type. The children of a q-node form a contiguous interval, but can only be in the given order or its reverse.

[1] Angluin et al. [8] have a small error in their argument because their potential function fails to explicitly consider the number of p-nodes, which creates a problem for some of the PQ-tree updates. We fix this, without affecting their asymptotic bound. For the ordered constraints case, we are also able to obtain a much finer analysis.

Every time a new interval constraint comes, the tree updates itself by identifying any of the eleven patterns, P0, P1,..., P6, and Q0, Q1, Q2, Q3, of the arrangement of nodes and replacing it with each correspondent replacement. The update fails when it cannot identify any of the patterns, in which case the contiguous intervals fail to produce any consistent permutation. We refer readers to Section 2 of Booth and Lueker [11] for a more detailed description of PQ-trees.

The reason we can use a PQ-tree to guide our algorithm is because of an observation made in Section 1 that each ordered constraint $(v_1, v_2, v_3, \ldots, v_{k-1}, v_k)$ is equivalent to $k-1$ connectivity constraints $S_2, \ldots, S_k$, where $S_i = \{v_1, \ldots, v_i\}$. Note that each connectivity constraint corresponds to an interval in the underlying graph. So upon seeing one ordered constraints, we reduce the PQ-tree with the equivalent interval constraints, *in order*. Then what our algorithm does is simply to add edge(s) to the graph every time a pattern is identified and replaced with its replacement, so that the graph satisfies all the seen constraints. Note that to reduce the PQ-tree with one interval constraint, there may be multiple patterns identified and hence multiple edges may be added.

Before running into details of how the patterns determine which edge(s) to add, we note that, without loss of generality, we can assume that the algorithm is in either one of the following two stages.

- The PQ-tree is about to be reduced with $\{v_1, v_2\}$.
- The PQ-tree is about to be reduced with $\{v_1, \ldots, v_k\}$, when the reductions with $\{v_1, v_2\}$ $\cdots$, $\{v_1, \ldots, v_{k-1}\}$ have been done.

Because of the structure of constraints discussed above, we do not encounter all PQ-tree patterns in their full generality, but in the special forms demonstrated in Table 1. Based on this, we make three important observations which can be verified by carefully examining how a PQ-tree evolves along with our algorithm.

1. The only p-node that can have more than two children is the root.
2. At least one of the two children of a non-root p-node is a leaf node.
3. For all q-nodes, there must be at least one leaf node in any two adjacent children. Hence, Q3 doesn't appear.

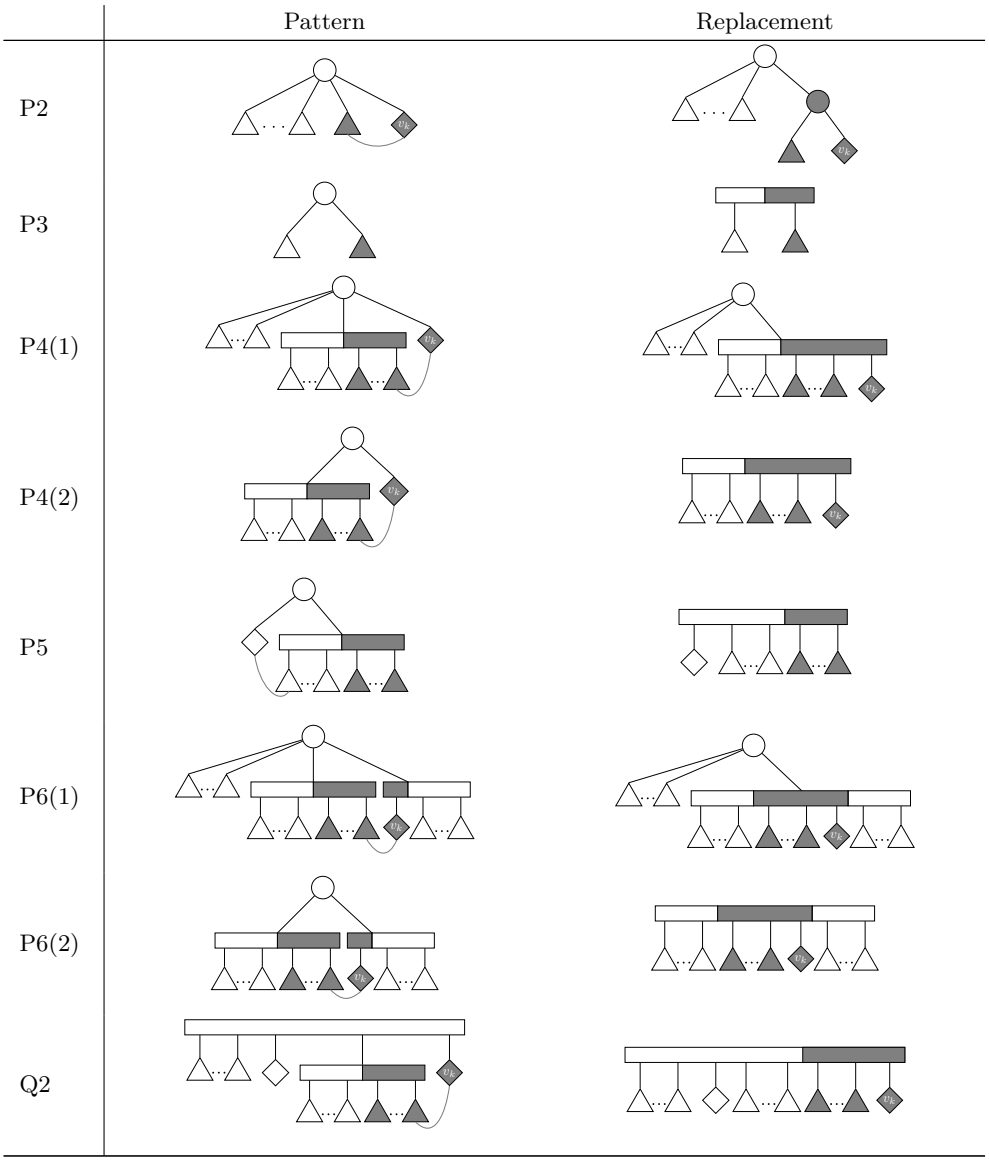
Now we describe how the edges are going to be added. Note that a PQ-tree inherently learns edges that appear in the optimum solution even when those edges are not forced by constraints. Apart from adding edges that are necessary to satisfy the constraints, our algorithm will also add any edge that the PQ-tree inherently learns. For all the patterns except Q2 such that a leaf node v_k is about to be added as a child to a different node, we can add one edge joining v_k to v_{k-1}. For all such patterns except Q2, it is obvious that this would satisfy the current constraint and all inherently learnt edges are also added. For Q2, the PQ-tree could learn two edges. The first edge is (v_k, v_{k-1}). The second one is an edge between the leftmost subtree of the daughter q-node (call T_l) and the node to its left (call v_l). Based on Observation 3, v_l is a leaf. But based on the algorithm, one of these two edges is already added. Hence, we only need to add one edge when Q2 is applied. For P5, we add the edge as shown in Table 1.

Let us denote by P and Q the sets of p-nodes and q-nodes, respectively, and by $c(p)$ the number of children node p has. And let potential function ϕ of a tree T be defined as

$$\phi(T) = a \sum_{p \in P} c(p) + b|P| + c|Q|,$$

where a, b, and c are coefficients to be determined later.

Table 1 Specific patterns and replacements that appear through the algorithm. P4(1) denotes the case of P4 where the top p-node is retained in the replacement and P4(2) denotes the case where the top p-node is deleted. The same is true for P6. P0, P1, Q0, and Q1 are just relabelling rules, and we have omitted them because no edges need to be added. We use the same shapes to represent p-nodes, q-nodes, and subtrees as in Booth and Lueker's paper [11] for easy reference, and we use diamonds to represent leaf nodes.



We want to upper bound the number of edges added for each pattern by the drop of potential function. We collect the change in the three terms in the potential function that each replacement causes in Table 3, and we can solve a simple linear system to get that choosing $a = 2$, $b = -3$, and $c = 1$ is sufficient. For ease of analysis, we add a dummy vertex v_{n+1} that does not appear in any constraint. Now, the potential function starts at $2n - 1$ (a single p-node with $n + 1$ children) and decreases to 2 when a path is uniquely

Table 3 How the terms in the potential function: $\sum_{p\in P} c(p)$, $|P|$, and $|Q|$ change according to the updates.

	$\sum_{p\in P} c(p)$	$\lvert P\rvert$	$\lvert Q\rvert$	$-\Delta\Phi$	number of edges added
P2	1	1	0	$-a-b$	1
P3	-2	-1	1	$2a+b-c$	0
P4(1)	-1	0	0	a	1
P4(2)	-2	-1	0	$2a+b$	1
P5	-2	-1	0	$2a+b$	1
P6(1)	-1	0	-1	$a+c$	1
P6(2)	-2	-1	-1	$2a+b+c$	1
Q2	0	0	-1	c	1
Q3	0	0	-2	$2c$	1

determined. Hence, the number of edges added by the algorithm is $2n-3$, which gives the desired asymptotic upper bound. ◀

References

1 Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics*, 18(4):697–712, 2005.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. doi:10.1137/060661946.

3 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Seffi Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006.

4 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing*, 33(2):487–501, 2004.

5 Dana Angluin, James Aspnes, and Lev Reyzin. Inferring social networks from outbreaks. In *International Conference on Algorithmic Learning Theory*, pages 104–118. Springer, 2010.

6 Dana Angluin, James Aspnes, and Lev Reyzin. Inferring social networks from outbreaks. In *ALT*, pages 104–118, 2010.

7 Dana Angluin, James Aspnes, and Lev Reyzin. Optimally learning social networks with activations and suppressions. *Theor. Comput. Sci.*, 411(29-30):2729–2740, 2010. doi:10.1016/j.tcs.2010.04.008.

8 Dana Angluin, James Aspnes, and Lev Reyzin. Network construction with subgraph connectivity constraints. *Journal of Combinatorial Optimization*, 29(2):418–432, 2015.

9 Dana Angluin and Jiang Chen. Learning a hidden graph using o (logn) queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008.

10 Richard Beigel, Noga Alon, Simon Kasif, Mehmet Serkan Apaydin, and Lance Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In *Proceedings of the fifth annual international conference on Computational biology*, pages 22–30. ACM, 2001.

11 Kellogg S Booth and George S Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

12 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal: dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

13 Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 109–118. ACM, 2007.

14 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

15 Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):21, 2012.

16 Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to dna physical mapping. *Discrete Applied Mathematics*, 88(1):147–165, 1998.

17 Anupam Gupta, Ravishankar Krishnaswamy, and R Ravi. Online and stochastic survivable network design. *SIAM Journal on Computing*, 41(6):1649–1672, 2012.

18 Ephraim Korach and Michal Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 98(1-3):385–414, 2003.

19 Ephraim Korach and Michal Stern. The complete optimal stars-clustering-tree problem. *Discrete Applied Mathematics*, 156(4):444–450, 2008.

20 Hervé Moulin and Francois Laigret. Equal-need sharing of a network under connectivity constraints. *Games and Economic Behavior*, 72(1):314–320, 2011. doi:10.1016/j.geb.2010.08.002.

21 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.

22 Lev Reyzin. *Active Learning of Interaction Networks*. PhD thesis, Yale University, New Haven, Connecticut, 2009.

23 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *International Conference on Algorithmic Learning Theory*, pages 285–297. Springer, 2007.

24 Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 67–75. Springer, 2008.

Complexity of Model Checking MDPs against LTL Specifications

Dileep Kini[*1] **and Mahesh Viswanathan**[†2]

1 **Akuna Capital LLC, Chicago, USA**
dileeprkini@gmail.com
2 **University of Illinois, Urbana-Champaign, USA**
vmahesh@illinois.edu

Abstract

Given a Markov Decision Process (MDP) $\mathcal{M}$, an LTL formula φ, and a threshold $\theta \in [0,1]$, the verification question is to determine if there is a scheduler with respect to which the executions of $\mathcal{M}$ satisfying φ have probability greater than (or $\geq$) θ. When $\theta = 0$, we call it the qualitative verification problem, and when $\theta \in (0,1]$, we call it the quantitative verification problem. In this paper we study the precise complexity of these problems when the specification is constrained to be in different fragments of LTL.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Markov Decision Processes, Linear Temporal Logic, model checking, complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.35

1 Introduction

Systems exhibiting both non-deterministic and probabilistic behaviors are semantically interpreted using Markov Decision Processes (MDPs) [9, 11, 4]. Markov Decision Processes are interpreted with respect to a scheduler who resolves the non-determinism at each step – a single step of the MDP has two phases, where the scheduler first picks a probabilistic transition out of the current state based on the sequence of states visited in the computation, and then a dice is rolled to stochastically choose the next state according to the transition chosen by the scheduler. The verification problem for MDPs with respect to specifications in LTL is as follows. Given an MDP $\mathcal{M}$, a formula φ, a threshold $\theta \in [0,1]$, determine if there is a scheduler with respect to which the measure of executions satisfying φ is greater than (or greater than or equal to) the threshold θ. A special case of this problem is when $\theta = 0$ which is called the *qualitative verification problem*. When $\theta \neq 0$, this is called the *quantitative verification problem*.

The standard approach to solving the verification problem is using the *automata theoretic method* [11, 4]. Here one translates the specification φ into a *deterministic* automaton $\mathcal{A}$, takes the cross product of $\mathcal{A}$ with the MDP $\mathcal{M}$ to construct a new MDP $\mathcal{M}'$, and then analyzes $\mathcal{M}'$ to check the desired property. The complexity of this procedure is polynomial in the size of the final MDP $\mathcal{M}'$. Since any LTL formula can be translated into a deterministic automaton of doubly exponential size, this approach shows that the verification problem

* This work was partly carried out while Dileep Kini was at the University of Illinois, Urbana-Champaign. Dileep Kini was partly supported by NSF award CNS-1314485.
† Mahesh Viswanathan was partly supported by NSF awards CNS-1329991 and CCF-1422798.

Table 1 Summary of results for quantitative and qualitative model checking of MDPs against various fragments. The upper bounds for results marked a follow from the standard translation of LTL to deterministic automata. Upper bounds for results marked b follow from the translation of LTL to limit deterministic automata in [6]. Finally, upper bounds for the result marked c follow from the translation of this fragment to deterministic automata presented in [1].

	Quantitative	Qualitative
$\mathcal{B}(L_{\Diamond,\wedge})$	**PSPACE**-complete	**NP**-complete
$\mathcal{B}(L_{\Diamond,\wedge,\vee})$	**EXPSPACE**-complete	
$\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$	**2EXPTIME**-complete^a	
$\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$	**EXPTIME**-complete^c	**EXPTIME**-complete^b
$\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee})$	**EXPSPACE**-complete	
$\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$	**2EXPTIME**-complete^a	

for MDPs is in **2EXPTIME** [11, 4]. One can prove a matching lower bound [4] which establishes the problem to be **2EXPTIME**-complete.

In a series of recent papers [5, 10, 6], the qualitative verification problem for MDPs has been investigated carefully. In particular, it has been shown that for an expressive fragment of LTL called LTL_D, the qualitative model checking problem is in **EXPTIME** (as opposed **2EXPTIME**). The basis of this result is an improved translation from LTL to a special class of automata called *limit deterministic automata* which are then used in the automata theoretic approach to verify the MDP. It is shown in [6], that the translation yields exponential sized automata for the fragment LTL_D which gives the improved upper bound.

In this paper, we continue this line of research to obtain a more complete picture about quantitative and qualitative verification of MDPs against fragments of LTL. In this endeavour, we are also inspired by [1, 7, 2] that characterize the complexity of solving 2-player games against objectives described using fragments of LTL. Consider LTL to be formulae in negation normal form built using Boolean operations $\vee, \wedge$ and temporal operators $\bigcirc$ (next), $\Diamond$ (eventually), $\Box$ (always), and $\mathcal{U}$ (until). Taking $L_{op_1,\ldots,op_k}$ to denote the LTL fragment consisting of formulae built only using the operators $op_1, \ldots, op_k$, and $\mathcal{B}(L_{op_1,\ldots,op_k})$ to be all boolean combinations of formulae in $L_{op_1,\ldots,op_k}$, our results are summarized in Table 1.

We begin by discussing our results for quantitative model checking. The upper bounds that pertain to time complexity classes (namely those marked a or c in Table 1) are obtained simply from the fact that these fragments can be translated into deterministic automata of exponential (for the result marked c) or doubly-exponential (for results marked a) size. For the other upper bounds in Table 1, we present a new space efficient algorithm to compute the probability of repeatedly visiting a set of states in Markov chains of *small diameter*; this result mimics a similar result in [1] for solving games on graphs with small diameter. The upper bounds are then obtained by translating the LTL fragment into deterministic Büchi automata of small diameter (using observations in [1]), taking the cross product with the MDP, guessing an optimal scheduler, and computing the probability of repeated reachability in the resulting Markov chain using the space efficient algorithm. The lower bounds are obtained by observing that the reductions in [1, 2] work in this case, when the universal player is replaced by a stochastic player. It is worth noting that our lower bounds apply to the special case when the threshold $\theta = 1$; thus, the difficulty in solving the quantitative verification problem does not stem from the numbers involved.

For qualitative verification, the upper bound of **EXPTIME** (marked b) follows from the results in [6]. The improved upper bound of **NP** for the fragment $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ is obtained

by refining the translation given in [5]. The new modified translation constructs a limit deterministic automata that is a disjoint union of exponentially many, polynomial sized deterministic automata. The **NP** algorithm then guesses one of these disjoint automata and analyzes the MDP relative to the guessed deterministic automaton.

2 Preliminaries

2.1 Strings and Prefixes

Given a set S, we use S^* denote the set of all finite sequences (finite words) of elements from S, and S^+ to denote all non-empty finite sequences over S. The length of a finite word u is denoted by $|u|$. We use S^ω to denote all infinite sequences over S. Given a (finite or infinite) word, we use u_i to denote i^{th} symbol in the sequence u (we assume indices start at 0), $u_{[0,i]}$ to denote the prefix $u_0u_1 \dots u_{i-1}$ of length i, and $u_{[i,\infty]}$ to denote the suffix $u_iu_{i+1} \dots$ starting at index i. For $u \in S^+$, we use $\langle u \rangle$ to denote the last element in the sequence u. Given an infinite word u, we use $\mathsf{inf}(u)$ to denote elements of S that appear infinitely often in u. We use $S^!$ to denote words in S^+ with distinct elements. The binary relations $<, \leq$ on S^* denote the prefix relations: $u < v$ iff u is a proper prefix of v. We have $u \leq v$ iff $u < v$ or $u = v$. We use $\lessdot$ to denote the covering relation of prefixes, i.e., $u \lessdot v$ iff $v = ua$ for some $a \in S$. A set $U \subseteq S^+$ is said to be closed under prefixes iff every non-empty prefix of a word in U is also in U.

▶ **Definition 1.** A *prefix tree* on a set S is a pair (V, r), such that $V \subseteq S^+$, the set of vertices, is closed under prefixes, and r is the unique element in V of length 1. A vertex $v \in V$ is called a *leaf* if there is no $u \in V$ for which $v < u$. The set of all leaf vertices of V is denoted by $\mathsf{Leaf}(V)$ and the set of all non-leaf vertices are called inner vertices, denoted by $\mathsf{Inner}(V)$. A prefix tree is infinite if V is infinite.

2.2 Linear Temporal Logic

We recall definitions related to LTL and its fragments. Let AP be a set of atomic propositions, and let Π denote state predicates which represent boolean formulae over AP. LTL formulae are constructed using state predicates from Π, boolean connectives *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), temporal connectives *always* ($\Box$), *eventually* ($\Diamond$), *until* ($\mathcal{U}$) and *next* ($\bigcirc$).

▶ **Definition 2** (LTL Syntax)**.** Formulae in LTL are given by the following syntax:

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\,\mathcal{U}\,\varphi \qquad\qquad p \in \Pi$$

We consider the usual semantics for LTL as given by [8]. In this paper we will be interested in different fragments of LTL. We denote by $L_{op_1,\dots,op_k}$ the set of formulae built using (only) the operators $op_1, \dots, op_k$. For a collection of formulae Γ, we use $\mathcal{B}(\Gamma)$ to denote all possible boolean combinations of formulae in Γ.

2.3 Markov Chains

A Markov chain M is a tuple (Q, δ, q_0) where Q is the set of states, $q_0 \in Q$ is the initial state and $\delta : Q \times Q \to [0, 1]$ is the probabilistic transition function where $\sum_{s' \in Q} \delta(s, s') = 1$ for all $s \in Q$. A labeling of a Markov chain is a function $L : Q \to 2^{AP}$ that maps each state to an assignment over the propositions AP.

A Markov chain (Q, δ, q_0) induces an underlying graph where each state in Q is a vertex and there is an edge from s to s' iff $\delta(s, s') > 0$. A (finite/infinite) path in the Markov chain is an finite/infinite path $\pi = q_0 q_1 q_2 \dots$ in the underlying graph starting at q_0. For such a path π, its trace under labeling L is defined as the sequence of assignments $tr(\pi) = L(q_0)L(q_1)L(q_2)\cdots$. Let $\mathsf{Paths}(M)$ denote the set of all infinite paths and $\mathsf{Paths_f}(M)$ the set of all finite paths in M starting from q_0. A Markov chain M induces a probability distribution $\Pr_M$ on the infinite paths of the Markov chain. We refer the reader to [3] for detailed definitions. For an LTL formula φ over propositions AP and labeling $L : Q \to 2^{AP}$, the set of paths of M that yield a trace in $[\![\varphi]\!]$ is measurable, i.e., the quantity $\Pr_M(\{\pi \in Paths(M) \mid tr(\pi) \in [\![\varphi]\!]\})$ is well defined. We abuse notation and simply write the above quantity as $\Pr_M([\![\varphi]\!])$.

A temporal property of particular interest is what is called *repeated reachability*. For a set of states $B \subseteq Q$ we use the LTL-like notation $\Box\Diamond B$ to denote paths that visit some state in B infinitely often, i.e., $\{\pi \in Paths(M) \mid \mathsf{inf}(\pi) \cap B \neq \emptyset\}$. The computation of $\Pr_M(\Box\Diamond B)$ requires familiarity with the structure of the underlying graph of M. A set of vertices of a directed graph are called strongly connected if every pair of vertices have paths to each other. A Strongly Connected Component (SCC) is a set of vertices S that is maximally strongly connected, i.e., no superset of S is strongly connected. The SCCs of a graph induces a directed acyclic graph where the vertices are the SCCs and there is an edge from one SCC to another if there is an edge going from a vertex in the first to a vertex in the second. A SCC is called bottom (BSCC) if there is no other SCC that can be reached from it. It is well known (see Chapter 10 of [3]) that a (infinite) path of a Markov chain almost certainly (i.e. with probability 1) ends up in one of the BSCCs and visits each of the vertices in that BSCC infinitely often. In order to compute $\Pr_M(\Box\Diamond B)$ it suffices to compute the probability of reaching BSCCs that have at least one state from B. We will build upon these ideas in our proofs.

2.4 Markov Decision Processes

A Markov decision process (MDP) $\mathcal{M}$ is a tuple (Q, Act, Δ, q_0) where Q is the set of states, Act is the set of actions, and q_0 is the initial state and $\Delta : Q \times Act \times Q \to [0, 1]$ is the probabilistic transition function where $\sum_{q' \in Q} \Delta(q, a, q') = 1$ for every $q \in Q$ and $a \in Act$. A labeling of a MDP is a function $L : Q \to 2^{AP}$ that maps each state to an assignment over the propositions AP.

A MDP executes as follows: it begins at state q_0 and non-deterministically picks an action $a_0 \in Act$. This is followed by stochastically choosing a state q_1 with probability $\Delta(q_0, a_0, q_1)$. This process is now continued with q_1 which gives us an infinite run of the form $q_0 a_0 q_1 a_1 q_2 \dots$. Note that an MDP includes non-determinism in the form of the action to be picked which is absent in Markov chains. Markov chains are special cases of MDPs, which are devoid of non-determinism. In order to define the probability measure in an MDP, one requires a scheduler (or adversary) that resolves this non-determinism. A scheduler $\mathfrak{S} : Q^+ \to Dist(Act)$ is a function that maps a sequence of states (states visited until a certain point) to a distribution on actions. The action is picked stochastically according to the distribution. Pure strategies $\mathfrak{S} : Q^+ \to Act$ are those where the distribution corresponds to picking a single action with probability 1. For the problems studied here pure schedulers suffice and therefore we restrict our attention to them. A scheduler $\mathfrak{S}$ *induces* a Markov chain $\mathcal{M}_{\mathfrak{S}} = (Q^+, \delta, q_0)$ where $\delta(u, v) = \Delta(\langle u \rangle, \mathfrak{S}(u), \langle v \rangle)$ if $u \lessdot v$ and 0 otherwise. The Markov chain $\mathcal{M}_{\mathfrak{S}}$ is then used to define the measure on sets of paths of $\mathcal{M}$. The probability measure of an event E under scheduler $\mathfrak{S}$ for MDP $\mathcal{M}$, denoted by $\Pr^{\mathfrak{S}}_{\mathcal{M}}(E)$ is defined as the measure $\Pr_{\mathcal{M}_{\mathfrak{S}}}(E)$ associated with event E in Markov chain $\mathcal{M}_{\mathfrak{S}}$. A labeling $L : Q \to 2^{AP}$

for $\mathcal{M}$ can be extended to a labeling $L' : Q^+ \to 2^{AP}$ for $\mathcal{M}_{\mathfrak{S}}$ where $L'(u) = L(\langle u \rangle)$, which can then be used to define $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\varphi]\!])$ for LTL formula φ over propositions AP.

▶ **Definition 3.** The *quantitative* model checking problem for LTL is to decide if there exists a scheduler $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\varphi]\!]) \geq \theta$ given MDP $\mathcal{M}$, $\varphi \in$ LTL, and $\theta \in [0, 1]$ as inputs.

Two schedulers $\mathfrak{S}_1, \mathfrak{S}_2$ for MDP $\mathcal{M}$ are said to be equivalent, denoted by $\mathfrak{S}_1 \sim_{\mathcal{M}} \mathfrak{S}_2$, when $\mathsf{Paths}_{\mathsf{f}}(\mathcal{M}_{\mathfrak{S}_1}) = \mathsf{Paths}_{\mathsf{f}}(\mathcal{M}_{\mathfrak{S}_2})$ and $\mathfrak{S}_1(u) = \mathfrak{S}_2(u)$ for every $u \in \mathsf{Paths}_{\mathsf{f}}(\mathcal{M}_{\mathfrak{S}_1})$. Equivalent schedulers yield Markov chains whose reachable portions are isomorphic. All equivalent schedulers for a MDP can be viewed as tree which is obtained from "unfolding" the MDP under the scheduler. We define prefix trees associated with an MDP and then see how they are related to schedulers.

▶ **Definition 4.** Given a MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$, a *$\mathcal{M}$-labeled prefix tree* (V, q_0, λ), is one where (V, q_0) is a prefix tree on Q, and $\lambda : \mathsf{Inner}(V) \to Act$ is a labeling such that $\forall u \in \mathsf{Inner}(V), q \in Q : uq \in V$ iff $\Delta(\langle u \rangle, \lambda(u), q) > 0$.

Let $\mathcal{S}$ denote the set of all schedulers, and $\mathcal{S}/\!\sim_{\mathcal{M}}$ denote the equivalence classes induced by the $\sim_{\mathcal{M}}$ relation. The proposition below captures the observation that equivalent schedulers of $\mathcal{M}$ can be identified by their the infinite $\mathcal{M}$-labeled prefix tree obtained by unfolding $\mathcal{M}$ on those schedulers.

▶ **Proposition 5.** *Given MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$ there is a one to one correspondence between $\mathcal{S}/\!\sim_{\mathcal{M}}$ and infinite $\mathcal{M}$-labeled prefix trees (V, q_0, λ).*

3 Quantitative Model Checking

3.1 Upper Bounds

We will present upper bounds on the complexity of quantitative verification of MDPs against formulae from different LTL fragments. In the automata theoretic approach, if the LTL specification can be translated to a deterministic Büchi automaton, then the complexity is intrinsically tied to solving the problem of computing the optimal probability for repeatedly reaching a set of states. One of the contributions of this paper is a new space efficient algorithm for the repeated reachability problem in MDPs which is presented in Section 3.1.1. Before presenting this algorithm and using it to get the results in Table 1, we need to introduce two special classes of schedulers – *memoryless* and *depth-bounded* schedulers – and some simple observations about them.

▶ **Definition 6.** A memoryless scheduler $\mathfrak{S}$ is one where $\mathfrak{S}(u) = \mathfrak{S}(v)$ if $\langle u \rangle = \langle v \rangle$.

A memoryless scheduler uses only knowledge about the latest state to decide which action it is going to pick. For a finite execution u, $\langle u \rangle$ represents the latest state and hence the action $\mathfrak{S}(u)$ is only dependent on $\langle u \rangle$.

Next we define *depth-bounded schedulers* that generalize memoryless schedulers. Depth-bounded schedulers can make decisions based on the current history. However, they only consider the portion of history from which "loops" have been removed. For example, consider a history $u = q_1, q_2, q_3, q_4, q_2, q_5$. The sequence q_2, q_3, q_4, q_2 is a loop, and removing it from u gives the history $v = q_1, q_2, q_5$. The action chosen by a depth-bounded scheduler on history u is the same as the one chosen on history v. This is formally defined next.

▶ **Definition 7.** A depth-bounded scheduler $\mathfrak{S} : Q^+ \to Act$ is one such that

$$\forall v \in Q^*, u \in Q^!, m \in \{1, \ldots, |u|\} : \mathfrak{S}(uu_m v) = \mathfrak{S}(u_{[0,m]} v).$$

A depth-bounded scheduler removes loops from the current history as soon as they form, and makes its decision based on the truncated history. Note that the process of loop removal leaves the last state in the history unchanged. From this it follows that a memoryless scheduler is a special case of the depth-bounded scheduler where the decision depends only on $\langle u \rangle$.

▶ **Proposition 8.** *Every memoryless scheduler is a depth-bounded scheduler.*

Next, we define special kinds of prefix trees that correspond to depth-bounded schedulers. Let $\mathcal{D}$ denote all depth-bounded schedulers.

▶ **Definition 9.** A prefix tree (V, r) on S is called *depth-bounded* if V is finite, $\mathsf{Inner}(V) \subseteq S^!$ and $\mathsf{Leaf}(V) \cap S^! = \emptyset$.

Next, analogous to Proposition 5, we observe that there is a 1-to-1 onto correspondence between equivalence classes of depth-bounded scheduler for $\mathcal{M}$ and $\mathcal{M}$-labeled depth-bounded prefix trees. Let $\mathcal{D}$ denote all depth-bounded schedulers, and $\mathcal{D}/\!\sim_{\mathcal{M}}$ denote the equivalence classes induced by the $\sim_{\mathcal{M}}$ relation.

▶ **Proposition 10.** *Given MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$ there is a one to one correspondence between $\mathcal{D}/\!\sim_{\mathcal{M}}$ and $\mathcal{M}$-labeled depth-bounded prefix trees.*

3.1.1 Space efficient algorithm for repeated reachability

In the section, we present one of our core technical results, that gives a space-bounded algorithm for solving the quantitative repeated reachability problem for MDPs. The salient feature of the algorithm is that its space requirements are polynomial in the *diameter* of the underlying graph of the MDP and logarithmic in its size; here, by diameter we refer to the length of the longest simple path in the graph. Thus, this algorithm is space efficient for MDPs whose diameter is small when compared to its size.

▶ **Theorem 11.** *Consider MDP $\mathcal{M} = (Q, Act, \Delta, q_0)$ with diameter d, graph size n, and for any $q, q' \in Q$ and $a \in Act$, $\Delta(q, a, q')$ is a rational number of size at most k. Given a set of states $B \subseteq Q$, the problem of deciding if $\exists \mathfrak{S} : \Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B) \geq \theta$ can be solved in non-deterministic space $O(d^2 \cdot (\log(n) + k))$.*

The proof of the theorem relies on an algorithm that guesses a scheduler $\mathfrak{S}$, computes $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B)$ and compares it to θ. Recall that memoryless schedulers suffice for attaining the maximum probability of repeatedly reaching a set of states. Guessing a memoryless scheduler requires n bits of space, which does not meet our space requirements. But we know every memoryless scheduler is also a depth-bounded scheduler (Proposition 8), so it is suffices to look for a depth-bounded scheduler $\mathfrak{S}$. We use Proposition 10 to guess the $\mathcal{M}$-labeled depth-bounded prefix tree $T(\mathfrak{S}) = (V, q_0, \lambda)$. Storing the entire tree would use too much space; instead we guess the tree in a path-by-path manner using a depth first strategy (DFS). In this approach, at any given time, we only store a single path in the tree $T(\mathfrak{S})$. Observe that any single path in the depth-bounded tree has to be a path in $\mathcal{M}$ and hence bounded by the diameter d. Therefore storing a path and its labels requires only $d \cdot \log(n)$ bits of space. To complete the proof of Theorem 11, we need to describe how to compute the probability $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B)$ as we guess and explore the tree. The Markov chain $\mathcal{M}_{\mathfrak{S}}$ induced by a depth-bounded scheduler $\mathfrak{S}$ can be shown to be (probabilistic) bisimulation [3] equivalent to Markov chain $\mathcal{M}_{T(\mathfrak{S})}$, which is obtained from $T(\mathfrak{S})$ as follows: $\mathcal{M}_{T(\mathfrak{S})} = (\mathsf{Inner}(V), \delta, q_0)$

where

$$\delta(u,v) \stackrel{\text{def}}{=} \begin{cases} \Delta(\langle u\rangle, \lambda(u), \langle v\rangle) & \text{if } (u \lessdot v) \text{ OR } (v \leq u \text{ and } u\langle v\rangle \in \mathsf{Leaf}(V)) \\ 0 & \text{otherwise.} \end{cases}$$

We classify the edges of the above Markov chain as: (i) *forward edges* (u,v) where $u \lessdot v$, (ii) *back edges* (u,v) where $v \leq u$.

The probability of repeatedly reaching B in $\mathcal{M}_{\mathfrak{S}}$ equals the probability of repeatedly reaching B' in $\mathcal{M}_{T(\mathfrak{S})}$ where $B' = \{v \in \mathsf{Inner}(V) \mid \langle v\rangle \in B\}$. Computing repeated reachability in Markov chains boils down to computing probability of reaching BSCCs that contain at least one of the states in B'. In the remainder of this section we see how to do this during the DFS that explores the tree $T(\mathfrak{S})$.

Index Vertex of a SCC. For a SCC of $\mathcal{M}_{T(\mathfrak{S})}$ define an *index* vertex w as one for which there is no other vertex w' in the SCC such that $w' < w$. The first observation is that there is a unique index vertex for a SCC. For contradiction assume there are two indices $u \neq v$ for a SCC. By definition of SCC, there is a simple path from u to v. If this path does not contain any back edges then clearly $u < v$ contradicting the fact that v is an index. If the path contains back edges, consider the first back edge in the path, say (u', v'). Now v' cannot be on the path from u to u', due to the fact that nodes cannot be repeated on a simple path (otherwise v' would be visited twice from u to v). Now, $v' < u'$ (because back edges lead to a prefix/ancestor), and $u \not< v'$. This means $v' < u$ because two ancestors of any node in a tree are always directly related. Since there is a path from u to v' (as observed), and a path from v' to u owing to the fact that $v' < u$, we get that v' is included in the SCC. This contradicts u being an index. This proves the uniqueness of an index node. In our algorithm we guess which nodes in the tree are index nodes and compute the probability of reaching every index node. The probability of reaching a BSCC is simply the probability of reaching the index vertex of that BSCC.

Parent-Child relationship between index vertices. In order to compute the probability of reaching an index node in an inductive fashion, we identify the parent-child relationship between index vertices. An index node v is called the *child* of an index node u if $u < v$ and there is no index node w such that $u < w < v$. Similarly u is called the *parent* of v, if v is the child of u. Note that every index has a unique parent except for the root which has no parent. For an index node u let $C(u)$ denote all the children index nodes of u. Given a node u, let p_u denote the probability of reaching u. Given a node uv with $u, v \neq \epsilon$, let $q_{u,v}$ denote the probability of moving from u to uv along the unique path of forward edges from u to uv in $\mathcal{M}_{T(\mathfrak{S})}$. $q_{u,v}$ is given by

$$q_{u,v} \stackrel{\text{def}}{=} \prod_{i=0}^{|v|-1} \delta(u.v_{[0,i]}, u.v_{[0,i+1]}) \tag{1}$$

The Proposition below formulates how we can calculate the probability of reaching an index by using the reachability probability of its parent.

▶ **Proposition 12.** *For a node $uv \in C(u)$, the probability p_{uv} of reaching uv is given by $(p_u \cdot q_{u,v})/s_u$, where $s_u \stackrel{\text{def}}{=} \sum_{w \in C(u)} q_{u,w}$ is called the normalizing factor of u.*

In order to use the above formula for the computation of p_{uv} we need to know the normalizing factor s_u of the parent node u. We guess this quantity s_u associated with each

node u that is an index, and store it along with u on the depth-first stack. Now, let us see how these can be used to compute reachability probabilities. For an index node uv whose parent is u, assume p_u is already computed and stored. The parent of a node can be identified by looking at the latest node before u in the stack that is an index node. For the root node r, $p_r = 1$. Now p_{uv} can be computed according to Proposition 12 using:

- p_u which is already computed and stored on the stack when u was first encountered
- s_u which is guessed and stored on the stack when u was first encountered
- $q_{u,v}$ which can be computed by looking at the path from u to uv on the depth-first stack.

Next, to compute the probabilities of reaching the BSCCs that have a state from B' in them, we observe that an index node u corresponds to a BSCC iff the normalizing factor $s_u = 0$, implying that it has no children. For such a state u we mark it as *final* as soon as a descendant uv is encountered where $\langle uv \rangle \in B$. Once all the descendants of u are explored we check if it is final, and if so we add the probability of reaching it, p_u, to a running total. The total value at the end of the DFS exploration is the required probability $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Box\Diamond B)$.

Confirming guesses. In the computation described above we have guessed two things for every node u: (a) if u is an index or not; (b) the normalizing factor s_u, whenever u is an index. In order to check that our guess regarding u being an index is correct we use the following:

▶ **Proposition 13.** *For $T(\mathfrak{S}) = (V, q_0, \lambda)$, a node $u \in \mathsf{Inner}(V)$ is an index node of some SCC in $\mathcal{M}_{T(\mathfrak{S})}$ iff every $uv \in \mathsf{Leaf}(V)$ is such that $\langle uv \rangle = \langle uv' \rangle$ for some $uv' \in \mathsf{Inner}(V)$.*

So, when u is guessed as an index node we make sure that every leaf descendant of u points to a repetition of a state that is no earlier than u, and when u is guessed as non-index we ensure there is a leaf descendant of u pointing to a repetition of a state earlier than u. In order to check that the guess for s_u is correct, we maintain a running sum for each index node u on the stack. When a $uv \in C(u)$ is encountered we add the computed quantity $q_{u,v}$ to the running sum associated with u. When the DFS exploration for u is complete we check that the running sum equals the guess s_u.

Size of numerical quantities. So far we have not accounted for the space requirements of the quantities we calculate. Let us begin by looking at $q_{u,v}$ for parent-child indices u, v. Equation 1 tells us that $q_{u,v}$ is a product of transition probabilities from u to v of which there are at most d. Therefore $q_{u,v}$ requires $d{\cdot}k$ bits to store since each transition probability has no more than k bits. Next, the normalizing factor s_u for an index u is the sum of $q_{u,v}$ where uv is a child of u. Note that the number of children for any index is bounded by the total number of nodes in the tree which is at most n^d. So each s_u, the sum of n^d quantities $(q_{u,v})$ each of size $d{\cdot}k$ requires only $d{\cdot}(\log(n) + k)$ bits. By Proposition 12, p_u is $p_{u'} \cdot q_{u',v'}/s_{u'}$, where u' is the parent of u and $u = u'v'$. By induction on the number of ancestors of u, we can argue that p_u has at most $O(d^2 \cdot (\log(n) + k))$ bits, since the maximum number of ancestors for any index node is d. The sum of p_u for u that are final BSCCs of which there are no more than n^d, will increase the bits required by $d{\cdot}\log(n)$. So the total space required remains $O(d^2 \cdot (\log(n) + k))$.

3.1.2 Upper Bounds for LTL-fragments

We are now ready to present all our upper bounds for the quantitative verification problem for different LTL fragments. Our results rely on the automata theoretic approach that solves

the quantitative verification problem by constructing a deterministic automaton for the given LTL specification. We, therefore, begin by recalling results on translations of fragments of LTL to deterministic automata.

▶ **Theorem 14** (Alur-LaTorre [1]). *The following fragments of* LTL *can be translated into deterministic Büchi automata with the following space and diameter bounds.*

- $L_{\Diamond,\wedge}$ *has automata of exponential size and linear diameter.*
- $L_{\Diamond,\bigcirc,\wedge}$ *has automata of exponential size and exponential diameter.*
- $L_{\Diamond,\wedge,\vee}$ *has automata of double exponential size and exponential diameter.*
- $L_{\Diamond,\bigcirc,\wedge,\vee}$ *has automata of double exponential size and exponential diameter.*
- $L_{\Diamond,\Box,\wedge,\vee}$ *has automata of double exponential size and double exponential diameter.*

These bounds on size and diameter are also tight.

We now present our upper bound results for quantitative verification shown in Table 1.

▶ **Theorem 15.** *The quantitative verification problem for MDPs against* LTL *specifications has the following complexity bounds – for* $\mathcal{B}(L_{\Diamond,\wedge})$ *it is in* **PSPACE***; for* $\mathcal{B}(L_{\Diamond,\wedge,\vee})$ *it is in* **EXPSPACE***; for* $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ *it is in* **2EXPTIME***; for* $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ *it is in* **EXPTIME***; for* $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee})$ *it is in* **EXPSPACE***; for* $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ *it is in* **2EXPTIME**.

Proof Sketch. Recall that in the automata theoretic approach to quantitative verification of MDPs, the LTL specification φ is translated into a deterministic automaton $\mathcal{A}$, and then the cross product of $\mathcal{A}$ with the MDP $\mathcal{M}$ is analyzed. When the automaton $\mathcal{A}$ is Büchi , the analysis involves solving the repeated reachability problem on the cross product MDP. The algorithm of [11, 4] runs in time that is polynomial in the size of the cross product. Given the results on the size of deterministic Büchi automata mentioned in Theorem 14, we immediately get the complexity bounds for $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$, $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$, and $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$.

For the other upper bounds, we follow a similar approach, but we construct the product of $\mathcal{M}$ and $\mathcal{A}$ on the fly. We exploit the fact that the Büchi automata constructions for LTL formulae have a representation that allows one to guess its states and check the transition relation just from knowing the formula. This allows us to apply Theorem 11 to the implicit product whose diameter is the product of the diameters of $\mathcal{M}$ and $\mathcal{A}$. Given the bounds on the diameter of the deterministic Büchi automata mentioned in Theorem 14, and using Theorem 11, we obtain the complexity bounds for the remaining fragments. ◀

3.2 Lower Bounds

In this section we prove matching lower bounds for the upper bounds established in Theorem 15. The lower bounds essentially follow from lower bounds established in [1, 2] for 2-player games. The reason for this observation is that games constructed in the lower bound reductions in [1, 2] have a special property that enable their lifting to the quantitative verification problem for MDPs. We begin this section by identifying this property, and showing how it helps transfer complexity bounds to the quantitative verification case.

Recall that a two player game is played on a graph $G = (V, E)$, where the set of vertices V is partitioned into two sets – $V_\exists$ which belong to $\exists$-player, and $V_\forall$ which belong to $\forall$-player. At any given time, the play is at some vertex u of graph G. Player P ($P \in \{\exists, \forall\}$) plays from u if $u \in V_P$, by picking the target of some outgoing edge from u. Starting from an initial vertex u_0, a *play* is the infinite sequence of vertices visited as the players choose edges on their turn. Given an objective described by LTL formula φ, we say a play π is winning for $\exists$-player if π satisfies φ; otherwise the play is said to be winning for the $\forall$-player. We now identify a special class of games that we call *finitely winnable*.

▶ **Definition 16.** A game (G, u_0, φ) is said to be *finitely-winnable* for a player P ($P \in \{\exists, \forall\}$) iff for any play π of (G, φ) that P wins, there is a prefix of π, say π', such that every play (according to game graph G) that is an extension of π' is also winning for P.

The main observation about games that are finitely winnable for the $\forall$-player is that if the $\forall$-player is replaced by a stochastic player that uniformly chooses among the available choices, then in the resulting MDP, there is a scheduler that meets objective φ with probability 1 if and only if the $\exists$-player has a winning strategy in the game.

▶ **Proposition 17.** *Given a game (G, u_0, φ) which is finitely-winnable for the $\forall$-player, the MDP $\mathcal{M}_G$ obtained by replacing the $\forall$-player with stochastic choices is such that the $\exists$-player has a winning strategy for (G, u_0, φ) if and only if there exists a scheduler $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) = 1$.*

Proof. If the $\exists$-player has a winning strategy for (G, u_0, φ), then the strategy interpreted as scheduler for $\mathcal{M}_G$ is going to be such that all runs of that scheduler are going to satisfy φ, which implies $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) = 1$. Now consider the case the where the $\forall$-player has a winning strategy for (G, u_0, φ). Here, for any strategy for the $\exists$-player, there is going to be a play that is won by the $\forall$-player. Since the game is finitely-winnable for the $\forall$-player, we know there is a prefix of the play whose every extension is winning for the $\forall$-player. What this means in the MDP setting, is that for any strategy there is a finite run ρ whose every extension results in φ not being met. Since the measure associated with all extensions of ρ is non-zero (since ρ is finite), we get that any strategy loses with non-zero probability, i.e., $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) < 1$ for any scheduler $\mathfrak{S}$. ◀

We use the above observations to obtain matching lower bounds for the quantitative verification problem.

▶ **Theorem 18.** *The quantitative verification problem for MDPs against* LTL *specification has the following complexity lower bounds – for $\mathcal{B}(L_{\Diamond,\wedge})$ it is* **PSPACE***-hard; for $\mathcal{B}(L_{\Diamond,\wedge,\vee})$ it is* **EXPSPACE***-hard; for $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ it is* **2EXPTIME***-hard; for $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ it is* **EXPTIME***-hard; for $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee})$ it is* **EXPSPACE***-hard; for $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ it is* **2EXPTIME***-hard.*

Proof Sketch. All the lower bounds follow from similar lower bounds established for solving 2-player games for the same LTL fragments in [1, 2]. All reductions for games essentially reduce the membership problem of a space/time bounded Alternating Turing machine (ATM) – given an ATM $\mathcal{A}$ and an input w they construct a game graph G, initial state u_0, and a specification φ such that $w \in L(\mathcal{A})$ iff there exists a winning strategy for the $\exists$-player in the game (G, u_0, φ). In each of these reductions, the game (G, u_0, φ) is finitely winnable for the $\forall$-player. Thus, we can use the same reduction and Proposition 17 to obtain a lower bound for the quantitative verification problem for MDPs when the threshold is $\theta = 1$. ◀

4 Qualitative Model Checking

The qualitative model checking problem is the following: given MDP $\mathcal{M}$ and LTL formula φ, check if there exists a scheduler $\mathfrak{S}$ under which the probability of paths satisfying φ is non-zero. In this section, we present results that refine our understanding of the complexity of qualitative verification for LTL fragments.

4.1 Upper Bounds

The qualitative problem for MDPs against LTL can be solved using an automata-theoretic approach described by [11, 4] in which the LTL formula is translated into a *limit-deterministic* automata. An automaton is said to be limit-deterministic (or deterministic in the limit) if every state reachable from a final state is deterministic. The result proved by [4] is as follows:

▶ **Proposition 19.** *Given an MDP $\mathcal{M}$ and a limit-deterministic Büchi automaton $\mathcal{A}$, the problem of checking if there exists $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\mathcal{A}]\!]) > 0$, can be solved by taking a cross-product of $\mathcal{M}$ and $\mathcal{A}$ and checking if this product has a reachable BSCC containing a state (s, q) where q is a final state of $\mathcal{A}$.*

Checking for the existence of such a *final* BSCC boils down to analyzing the product graph which runs in linear time. We have recently shown how to transform LTL to limit-deterministic Büchi automata (LDBA) such that the construction is of exponential size for a large class of properties, namely LTL_D [6]. This allows us to prove an **EXPTIME** upper bound for qualitative model checking against that fragment using the result above. In this section we see that the problem is in **NP** for the fragment $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$. The construction in [5] for $\varphi \in \mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ produces an exponential sized LDBA $\mathcal{A}_\varphi$, so it would seem Proposition 19 is not useful for proving our desired upper bound. The key idea we introduce here is the following: the automaton $\mathcal{A}_\varphi$ can be *split* into a disjoint union of exponentially many LDBAs each of which is polynomially large in the size of φ. In Proposition 19, if $\mathcal{A}$ is a disjoint union of multiple LDBAs, then the product of $\mathcal{M}$ and $\mathcal{A}$ has reachable final BSCC if and only if the product of $\mathcal{M}$ and some individual component of $\mathcal{A}$ has a final BSCC. The **NP**-algorithm guesses this individual component of $\mathcal{A}$ (of polynomial size) and analyzes the product graph of $\mathcal{M}$ and the individual component in time polynomial in both $\mathcal{M}$ and φ.

In order to understand the *splitting* of $\mathcal{A}_\varphi$ we recall the core idea behind the construction of $\mathcal{A}_\varphi$ for $\varphi \in \mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$. For each $\Diamond$ or $\Box$ subformula ψ of φ, the automaton keeps track of *how often ψ is true*, which is one of three things: (a) ψ is always true; (b) ψ is true at some point but not always; (c) ψ is never true. This yields a tri-partition, $\pi = \langle\, \alpha \,|\, \beta \,|\, \gamma \,\rangle$, of all the $\Diamond, \Box$ subformulae of φ. If a $\Diamond$ subformula is in α, β or γ we take it to mean that it is never true, true at some point but not always, or always true, respectively. Dually, if $\Box$ subformula is in α, β or γ we take it to mean that it is always true, true at some point but not always, or never true, respectively. With this semantics in mind we see that a subformula in α or γ should remain in α or γ respectively in the future, and a subformula in β can remain in β only for a finite time before moving to α. It turns out that, for a given input word w, correctly guessing (a) the triple π at the beginning of w and (b) the points along w at which subformulae move from β to α, enables us to check if w satisfies the original formula φ. A key observation here is that a triple at a certain point not only tells us how often a $\Diamond, \Box$ subformula is true from that point onwards in the future, but also whether or not the subformula is true at that point. In other words, a triple refines the truth of $\Diamond, \Box$ formulae. This observation is used to inductively check that the guessed triple is correct at every point. We encourage the reader to refer to [5, 6] for a detailed account of the construction. For the purposes of this paper it suffices to know that the state of the automaton for φ is of the form (π, k) where:

- π is a triple reflecting how often the $\Diamond, \Box$ subformulae are true on the remaining input.
- k is an integer counter no larger than $|\varphi|$, which is updated deterministically.

The transitions of the automaton allow moving from a state with $\pi = \langle\, \alpha \,|\, \beta \,|\, \gamma \,\rangle$ to a state with $\pi' = \langle\, \alpha' \,|\, \beta' \,|\, \gamma' \,\rangle$ only if $\alpha \subseteq \alpha'$, $\beta' \subseteq \beta$ and $\gamma = \gamma'$ ($\pi \sqsubseteq \pi'$ for short) in accordance with the semantics we associate with the triple. That is $\pi \sqsubseteq \pi'$ is a necessary condition for

a transition to move from π to π', i.e., subformulae in β are allowed to move α while the remaining stay put. In order to *split* this automaton into smaller components as anticipated earlier, we add restrictions to the order in which the formulae in β are moved to α. First, let us fix an initial triple $\pi = \langle\, \alpha_0 \,|\, \beta_0 \,|\, \gamma_0 \,\rangle$. Given π, consider a ranking function $\rho : \beta_0 \to \mathbb{N}$ whose range is allowed to be any consecutive set of positive integers starting from 1, i.e. $\{1, \ldots, n\}$. Given π and ρ we are going to define a component $\mathcal{A}_{(\pi,\rho)}$ of the original automaton $\mathcal{A}_\varphi$. We define the space of possible triples $\pi_i = \langle\, \alpha_i \,|\, \beta_i \,|\, \gamma_i \,\rangle$ for $i \in \{0, 1, \ldots, n\}$ as follows: $\alpha_i = \{\psi \in \beta_0 \mid f(\psi) \leq i\} \cup \alpha_0$; $\beta_i = \{\psi \in \beta_0 \mid f(\psi) > i\}$; $\gamma_i = \gamma_0$ The states of $\mathcal{A}_{(\pi,\rho)}$ are those states of $\mathcal{A}_\varphi$ where the triple is restricted to be some π_i as defined above. A transition τ, say $(\pi_i, m) \xrightarrow{\sigma} (\pi_j, n)$, is allowed in $\mathcal{A}_{(\pi,\rho)}$ iff τ is a valid transition in $\mathcal{A}_\varphi$ and either $j = i$ or $j = i + 1$. In $\mathcal{A}_{(\pi,\rho)}$ a transition is allowed to either keep the triple unchanged (when $j = i$), or move only the formulae mapped to $i + 1$ from β to α (when $j = i + 1$). Thus the ranking function ρ restricts the order in which the subformulae move from β to α. A subformula with smaller rank is moved earlier compared to one with a larger rank. Note that two or more formulae can be mapped to the same number, which means those formulae are moved simultaneously. The initial state of $\mathcal{A}_{(\pi,\rho)}$ is defined to be $(\pi_0, 0)$ and the state $(\pi_n, 0)$ is marked as the only final state. Note that the size of the automaton $\mathcal{A}_{(\pi,\rho)}$ is $n + |\gamma_0|$ which is linear in $|\varphi|$. The number of different (π, ρ) is exponential in φ, hence there can be exponentially many different individual components.

What remains to be seen is that the disjoint union of these components $\biguplus \mathcal{A}_{(\pi,\rho)}$ accepts exactly the same language as $\mathcal{A}_\varphi$. Since $\mathcal{A}_{(\pi,\rho)}$ is a projection of $\mathcal{A}_\varphi$ it is the case that $[\![\mathcal{A}_{(\pi,\rho)}]\!] \subseteq [\![\mathcal{A}_\varphi]\!]$ and so $[\![\uplus \mathcal{A}_{(\pi,\rho)}]\!] \subseteq [\![\mathcal{A}_\varphi]\!]$. To see the other direction consider any word w accepted by $\mathcal{A}_\varphi$, and let $(\pi_0, k_0), (\pi_1, k_1), \ldots$ be an accepting run for w on $\mathcal{A}_\varphi$ with $\pi_i = \langle\, \alpha_i \,|\, \beta_i \,|\, \gamma_i \,\rangle$. From the construction of $\mathcal{A}_\varphi$ we know that $\pi_0 \sqsubseteq \pi_1 \sqsubseteq \pi_2 \cdots$. Identify all the positions $j_1 < j_2 < \cdots < j_n$ where the triple changes, i.e.,

$$(\pi_0 = \pi_1 \cdots = \pi_{j_1}) \sqsubset (\pi_{j_1+1} = \cdots = \pi_{j_2}) \sqsubset (\pi_{j_2+1} = \cdots = \pi_{j_3}) \sqsubset (\cdots) \sqsubset (\pi_{j_n} = \cdots$$

Here j_i is the i^{th} time the triple changes, n being the last. Now we consider the automaton $\mathcal{A}_{(\pi_0,\rho)}$ where $\rho(\psi) \stackrel{\text{def}}{=} i$ if ψ moves from β to α at position j_i, i.e., $\psi \in \beta_{j_i}$ and $\psi \in \alpha_{j_i+1}$. Observe that the above accepting run is also an accepting run of $\mathcal{A}_{(\pi,\rho)}$ on the word w. This gives us $[\![\mathcal{A}_\varphi]\!] \subseteq [\![\uplus \mathcal{A}_{(\pi,\rho)}]\!]$.

Thus we have successfully split $\mathcal{A}_\varphi$ into exponentially many individual components of linear size. The index (π, ρ) for any component requires only polynomially many bits to represent. This combined with our earlier observation of using Proposition 19 for the disjoint union gives us the **NP**-algorithm for qualitative model checking against $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$. We end this section by summarizing the upper bound results for qualitative model checking.

▶ **Theorem 20.** *The qualitative verification problem for MDPs against specifications in* $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ *is in* **NP** *and against specifications in* $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ *is in* **EXPTIME**.

Proof. The argument for qualitative model checking of $\mathcal{B}(L_{\Diamond,\Box,\wedge,\vee})$ being in **NP** has been spelt out above. The qualitative model checking problem of $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$ is in **EXPTIME** because $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$-formulae can be translated into exponential sized limit deterministic automata [5, 6] and the observation in Proposition 19. ◀

4.2 Lower Bounds

In this section, we show that the upper bounds proved in Theorem 20 are tight.

▶ **Theorem 21.** *The qualitative verification problem for MDPs against specifications in* $\mathcal{B}(L_{\Diamond,\wedge})$ *is* **NP***-hard and against specifications in* $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ *is in* **EXPTIME***-hard.*

We note that the **EXPTIME**-hardness for the fragment $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge})$ strengthens the result in [5] that establishes the hardness for the larger fragment $\mathcal{B}(L_{\Diamond,\Box,\bigcirc,\wedge,\vee})$.

5 Conclusions

In this paper, we presented results for the quantitative and qualitative verification problems for MDPs against fragments of LTL studied in [1, 2]. In doing so we refined the upper and lower bounds for qualitative verification that were obtained in [5, 10, 6].

Acknowledgements. We'd like to thank annonymous referees for their comments which improved the draft.

References

1 R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Logic*, 5(1):1–25, 2004.
2 R. Alur, S. La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *Proceedings of the International Conference on Concurrency Theory*, pages 128–143, 2003.
3 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.
5 Dileep Kini and Mahesh Viswanathan. Limit deterministic and probabilistic automata for LTL \GU. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 628–642, 2015.
6 Dileep Kini and Mahesh Viswanathan. Optimal translation of LTL to limit deterministic automata. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 113–129, 2017.
7 J. Marcinkowski and T. Truderung. Optimal complexity bounds for positive LTL games. In *Proceedings of the International Conference on Computer Science Logic*, pages 262–275, 2002.
8 A. Pnueli. The temporal logic of programs. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.
9 M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.
10 Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínskỳ. Limit-deterministic Büchi automata for linear temporal logic. In *Proceedings of the International Conference on Computer-Aided Verification*, pages 312–332, 2016.
11 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE Computer Society, 1985.

A Unified Method for Placing Problems in Polylogarithmic Depth*

Andreas Krebs[1], Nutan Limaye[2], and Michael Ludwig[3]

1 **University of Tübingen, Germany**
2 **IIT Bombay, India**
3 **University of Tübingen, Germany**

Abstract

In this work we consider the term evaluation problem which is, given a term over some algebra and a valid input to the term, computing the value of the term on that input. In contrast to previous methods we allow the algebra to be completely general and consider the problem of obtaining an efficient upper bound for this problem. Many variants of the problems where the algebra is well behaved have been studied. For example, the problem over the Boolean semiring or over the semiring $(\mathbb{N}, +, \times)$. We extend this line of work.

Our efficient term evaluation algorithm then serves as a tool for obtaining polylogarithmic depth upper bounds for various well-studied problems. To demonstrate the utility of our result we show new bounds and reprove known results for a large spectrum of problems. In particular, the applications of the algorithm we consider include (but are not restricted to) arithmetic formula evaluation, word problems for tree and visibly pushdown automata, and various problems related to bounded tree-width and clique-width graphs.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Polylogarithmic depth, Term evaluation, Parallel algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.36

1 Introduction

Background and motivation. Classically the notion of efficiently solvable problems is defined to be the class of problems for which there are polynomial time algorithms, namely the set of problems in the complexity class **P**. Over the last many decades a fine grained study of classically efficient computation has lead to many interesting subclasses of problems in **P**. One such class is a set of problems solvable using polynomially many processors which run in parallel for at most polylogarithmic time. This class of problems is known to be **NC**.

There are many interesting and fundamental computational problems for which the classical algorithms designed were inherently sequential in nature.

Owing to a series of theoretically intriguing and practically relevant discoveries we know **NC** algorithms for some of these problems. That is, a fairly diverse set of problems from the class **P** is known to be in **NC**. This raises a natural question: is **P** the same as **NC**? That is, can any sequential algorithm be turned into an efficient parallel one? In fact, this is one of the very central open questions in computer science. The question has implications to many different practical aspects of computer science such as distributed computing and parallel algorithms for large scale data (see for instance [25, 30, 38]).

* A full version of the paper is available at [22], `https://eccc.weizmann.ac.il/report/2017/019/`

A lot of effort has gone into understanding the relative strengths and weaknesses of **P** and **NC**. The study of Boolean and arithmetic circuits and many interesting results proved therein show that some specific subclasses of **NC** are strictly less powerful than **P**. (See for instance [13, 17, 18].) This rich body of work could be thought of as attempts to separate **NC** from **P**. On the other hand, over the last many years, surprising **NC** upper bounds have been proved for problems which were previously believed to be hard to parallelize. (See for instance [19, 28, 1, 16, 12].)

Our contributions. All these algorithmic advances raise a natural question: what makes a problem in **P** to have an **NC** algorithm? The main goal of this work is to build a theory which attempts to answer this question. Our main contributions are given below:

- We identify similarities between a large number of parallel algorithms. We observe that if a problem has a core *tree-like* structure, then it is amenable to have an **NC** algorithm. We formally define the notion of the tree-like structure and demonstrate the presence of such a structure in a large collection of problems.
- Our second important contribution is to mechanise the process of coming up with a parallel algorithm for any problem that has this tree-like structure. This can be thought of as an algorithmic contribution stemming from our work. We demonstrate the strength of this algorithmic technique by rediscovering many known **NC** upper bounds.
  This is also a technically challenging part of our work. The difficulty arises because we need an algorithm which is independent of the problem and only dependent on the underlying tree-like structure. The structure itself is dependent on the problem: for two seemingly unrelated problem these structures could be different. Therefore, the algorithm has to be general, which makes as few assumptions about the structure as possible.
  One caveat worth mentioning is that in some cases the tree-like structure is clear from the problem definition and in some cases it requires some work to notice this structure. We assume that an expert working on a specific problem may be able to notice this structure easily for the problem of her interest and then choose to use our approach mechanically to obtain an **NC** algorithm for the problem.

Significance. Over the last four decades there has been a lot of work related to design of parallel algorithms for tree-like problems. Given below is a notable and diverse (but not exhaustive) list of problems which have been considered in this literature.

- Boolean and arithmetic term evaluation [5, 7].
- Membership for language classes [26, 24, 10, 2, 21].
- Evaluation of circuits with bounded tree-width [20].
- Courcelle's Theorem and counting [9, 11].
- Computation of maximal cuts in bounded clique-width graphs [37].
- Counting Hamiltonian paths in bounded clique-width graphs [3, 37].

Using our techniques, we reprove the above results (see the full version [22]). That is, we give a unified way of proving all the above bounds. Moreover, we also consider variants of the above applications and obtain parallel ($\mathbf{NC}^1$, $\mathbf{NC}^2$, $\mathbf{SAC}^1$, $\mathbf{SAC}^2$) upper bounds.

Techniques. Our approach uses algebra as a tool for obtaining the desired abstraction. An algebra $\mathcal{A}$ consists of a set $\mathbb{D}$ and a finite set of operations $\{\oplus_1, \oplus_2, \ldots, \oplus_k\}$, where k is fixed. As mentioned earlier, we focus on coming up with an efficient parallel algorithm for problems with a tree-like structure. For this, we consider the *term evaluation problem*. A term is simply a tree in which the leaves are labelled by the elements of $\mathbb{D}$ and the internal nodes

are labelled by the operations. The term evaluation problem deals with evaluating the value of the term for a given assignment of the leaves from the domain $\mathbb{D}$. The main algorithmic and technical contribution of our work is to rewrite the term T as an equivalent term T' whose inputs are labelled by the elements of $\mathbb{D}$ and the internal nodes (gates) are labelled by the operations of an algebra $\mathcal{F}(\mathcal{A})$, which is an extended algebra derived from $\mathcal{A}$. Moreover, if T has size s then T' has size $\mathrm{poly}(s)$ and depth $O(\log s)$[1].

This result is our primary algorithmic contribution. It may be thought of as a meta theorem for the general term evaluation problem. Using this result we obtain **NC** upper bounds as follows: say Π is a problem for which we need to design an **NC** algorithm. For a given Π, we show that solving Π is equivalent to evaluating a term T_Π over an appropriately defined algebra $\mathcal{A}_\Pi$. Now, using the above result, we get a log-depth term over the operations of $\mathcal{F}(\mathcal{A}_\Pi)$. We then observe that each operation in $\mathcal{F}(\mathcal{A}_\Pi)$ is *easy* to evaluate. Note that for a given problem Π, there may be many $\mathcal{A}_\Pi$ one could design. It is not necessary for every choice of $\mathcal{A}_\Pi$, the corresponding $\mathcal{F}(\mathcal{A}_\Pi)$ has operations which are *easy* to evaluate. This part is sensitive to the choice of $\mathcal{A}_\Pi$. However, the main result stated above is independent of Π.

The known **NC** algorithms can be thought of as algorithms which transform T_Π to T'_Π for a particular Π. What we manage to do here is to obtain a transformation from T to T', irrespective of any specific Π (and hence a specific $\mathcal{A}_\Pi$). This approach allows us to unify many known results by noticing that each had to its core a term evaluation problem over an algebra. The following are the main technical contributions in this result: (i) the algebraic notions defined and used for the abstraction, and (ii) the definition of the extended algebra $\mathcal{F}(\mathcal{A})$. The notion of an extended algebra is intricate and crucial in the algorithm design.

Related work. Our algorithm for the term evaluation problem fits in the long chain of contributions dedicated to the term evaluation problem. The origin of which can be vaguely traced back to the investigation of upper bounds for the Boolean formula value problem. In [27] Lynch studied it first and achieved a log-space upper bound. Subsequently Cook conjectured that this bound is tight [8] which, as we know today, is not (unless log space equals log depth). Earlier, a way to deal with formulas that are very deep trees was investigated by Spira [33]: by a quadratic increase in size, we can balance a Boolean formula. Brent built upon this work [4]. Going from balancing to obtaining an **NC** (in fact $\mathbf{NC}^1$, i.e. log-depth) upper bound is not tough. It is known that if the transformation can be done in $\mathbf{NC}^1$, the evaluation is in $\mathbf{NC}^1$.

Cook and Gupta [15] as well as Ramachandran [31] were the next in line who showed that $\mathcal{O}(\log n \log\log n)$ deep circuits suffice for evaluating formulas. Based on [15], Buss showed an ALOGTIME bound [5] which equals logarithmic depth [32] and is known to be tight. His proof utilized a sophisticated two-player pebbling game. From there on the research proceeded in the direction of broadening the scope of the result. This continued research is always rooted in the work of [15] and [5]. Many other interesting works have contributed to this rich line of research, each solving the term evaluation problem over a specific algebra [29], [10], [23], [21].

Subsequent work. Closely related to our work is a very recent work of Ganardi and Lohrey [14] which uses the notion of algebras, terms and extended algebras (i.e. $\mathcal{A}, T, \mathcal{F}(\mathcal{A})$)

[1] Please refer to [36] for definitions of circuits, size and depth notions for terms and circuits and circuits with gates coming from an algebra. Also, the size can in fact be bounded by $O(s)$, but that is not crucial here.

and shows optimal upper bounds on the size of the **NC** circuits obtained for some of the problems considered here. This is an interesting piece of follow up work, which improves on the size of the circuits (to $O(n/\log n)$) for some of the problems and uses some of the machinery developed here.

Organization. We give the details regarding the term evaluation problem in Section 3. We present the relevant preliminaries and some definitions in Section 2. Finally the discussion regarding the applications of the term evaluation algorithm is provided in Section 4.

2 Preliminaries: notations and definitions

As mentioned before, the term evaluation problem that we deal with here is over a very general algebraic structure. In the literature, the term evaluation problem has been studied with respect to specific algebras. However, as our main goal here is to give a unified approach to solve the general term evaluation problem, we define algebraic structures which are as general as possible. We also define circuits (and terms) which *operate* over these algebraic structures and we formalize the notation of semantics for such circuits (and resp. terms).

To the best of our knowledge, the definitions appearing here have not been stated in this form in any other literature in the series of works related to the term evaluation problem. In that sense, they are new. However, some of the definitions are in fact generalizations/abstractions of well-known classical notions.

2.1 Notation

The set $\{1, \ldots, n\}$ is denoted by $[n]$ and $\{i, \ldots, j\}$ by $[i, j]$. The set $\mathbb{N}$ stands for the natural numbers containing 0, $\mathbb{Z}$ for the integers, and $\mathbb{B}$ for the Boolean values $\{\bot, \top\}$. An alphabet, denoted as Σ, is a finite sets of letters. A word $w \in \Sigma^*$ is a finite sequence of letters and hence Σ^* is the set of all words over Σ. The ith letter of w is denoted by $w(i)$. The length of w is denoted by $|w|$. The word of length 0 is denoted by ϵ. A language is a subset of Σ^*.

2.2 Many-sorted signatures, circuits and terms

Operations and sorts. Below we will deal with operations which get inputs from different domains. The distinct domains which any operation uses are called *sorts*. For example, consider an operation f which has four inputs, say x_1, x_2, x_3, x_4, the first two are Boolean, i.e. $x_1, x_2 \in \mathbb{B}$ and $x_3, x_4 \in \mathbb{N}$. The operation $f(x_1, x_2, x_3, x_4)$ outputs $x_3 \times x_4$ if x_1 AND x_2 is 1 and outputs $\frac{x_3}{x_4+1}$ otherwise (i.e. if x_1 AND x_2 is 0). Unlike a usual Boolean or arithmetic operation, f is more complicated. The inputs of f come from different sorts of domains, i.e. $\mathbb{B}$ and $\mathbb{N}$. The output is over yet another domain, namely $\mathbb{Q}$. Here, $\mathbb{B}, \mathbb{N}, \mathbb{Q}$ are the three different sorts.

We define, circuits, terms and algebras which use such generalized operations and also define the semantics for them. Towards this, we first define a many-sorted signature, which is a generalization of the notion of arity of a Boolean or arithmetic operation.

▶ **Definition 1** (Many-sorted signature)**.** Given $S \in \mathbb{N}$ different sorts, a many-sorted signature $\boldsymbol{\sigma}$ of an operation is an element of $[S]^* \times [S]$.

A many-sorted signature $\boldsymbol{\sigma}$ is a pair of a word and a letter (w, a). The word codes the input sorts of the operation. The length of a word $|w|$ is the arity of the operation. E.g. the operation f defined above has the signature $(1122, 3)$, where the three sorts are $\mathbb{B}, \mathbb{N}, \mathbb{Q}$

(numbered in that order). For a operation f with signature $\boldsymbol{\sigma}$, we write $\mathsf{In}_{\boldsymbol{\sigma}}(f)$ to denote w and $\mathsf{Out}_{\boldsymbol{\sigma}}(f)$ to denote a. If we have a set of operations $f_1, \ldots, f_\ell$, with signatures $\boldsymbol{\sigma_1}, \ldots, \boldsymbol{\sigma_\ell}$ respectively, we use σ to denote the tuple $(\boldsymbol{\sigma_1}, \ldots, \boldsymbol{\sigma_\ell})$ and $\sigma(i)$ to address (w_i, a_i). Also, $\mathsf{In}_\sigma(i, j)$ is the jth letter of $\mathsf{In}_\sigma(f_i)$. If we have an ordering on the operations $f_1, \ldots, f_\ell$ then we simply use $\mathsf{In}_\sigma(i)$ instead of $\mathsf{In}_\sigma(f_i)$. We use $|\sigma|$ to denote the number of different operations, i.e. ℓ.

A single-sorted signature $\boldsymbol{\sigma}$ is one where $|S| = 1$. In this case $\boldsymbol{\sigma}$ corresponds to the classical notion of signature assigning just an arity to the operation. For the sake of brevity, henceforth we will simply say signature instead of many-sorted signature.

We now define a circuit which uses these operations. Suppose a gate F in the circuit is assigned the operation f defined above. Then in such a circuit, one needs to ensure that the first two inputs to F are Boolean, while the last two are from $\mathbb{N}$. That is, only valid gate types feed into one another. We ensure this using the notion of a signature defined above. Formally, we define many-sorted circuits as follows.

▶ **Definition 2** (Many-sorted circuit)**.** For a signature σ, a many-sorted circuit over signature σ of S sorts, n inputs and m outputs is a tuple $C = (V, E, \mathsf{Order}, \mathsf{Gatetype}, \mathsf{Outputgates})$, where (V, E) is a directed acyclic graph, $\mathsf{Order}\colon E \to \mathbb{N}$ is an injective map giving an order on the edges, $\mathsf{Gatetype}\colon V \to [|\sigma|] \cup \{x_1, \ldots, x_n\}$ assigns a position of the signature or makes it an input gate, $\mathsf{Outputgates}\colon \{y_1, \ldots, y_m\} \to V$ makes gates output gates, such that:

- If some $v \in V$ has in-degree 0 then $\mathsf{Gatetype}(v) \in \{x_1, \ldots, x_n\}$ or $\mathsf{In}_\sigma(\mathsf{Gatetype}(v)) = \epsilon$, i.e. it is 0-ary.
- If some $v \in V$ has in-degree $k > 0$ then $|\mathsf{In}_\sigma(\mathsf{Gatetype}(v))| = k$, hence it is k-ary.
- For all $i \in [n]$ there exists at most one $v \in V$ such that $\mathsf{Gatetype}(v) = x_i$
- All successors of an input gate can be assigned a unique sort which is fixed by the successor gates and the signature. Also, $\mathsf{In}_C \subseteq [S]^n$ denotes the sorts of the n input gates and $\mathsf{Out}_C \subseteq [S]^m$ denotes the sorts of the m output gates.
- For all $v \in V$, let $v_1, \ldots v_{|\mathsf{In}_\sigma(\mathsf{Gatetype}(v))|}$ be the input gates for v such that $\mathsf{Order}(v_i) \leq \mathsf{Order}(v_j)$ iff $i \leq j$. Then $\mathsf{Out}_\sigma(\mathsf{Gatetype}(v_i)) = \mathsf{In}_\sigma(\mathsf{Gatetype}(v), i)$. If v_i is an input gate then $\mathsf{In}_C(j) = \mathsf{In}_\sigma(\mathsf{Gatetype}(v), i)$ where $\mathsf{Gatetype}(v_i) = x_j$.

By $\mathsf{Circ}_{\sigma,n,m}$ we denote the set of circuits over σ of n inputs and m outputs.

Terms and circuits are closely related. In general, a term is a circuit which is a tree. However in our setting additionally, a term has no input variables. Instead, the inputs are constants from the domain given as 0-ary operations. One can think of it as the case when all variables have already been assigned a value. Also our terms have only binary operations[2].

We also consider the notion of terms with an unknown. Such terms come into play when we decompose a given term in the term evaluation algorithm. Assume a term is to be evaluated over a domain $\mathbb{D}$ (e.g. $\mathbb{D}$ could be $\mathbb{B}, \mathbb{N}$ or $\mathbb{Q}$) then a term with an unknown evaluates to a map $\mathbb{D} \to \mathbb{D}$. Formally,

▶ **Definition 3** (Many-sorted term, many-sorted term with an unknown)**.** Given a many-sorted circuit T over σ where $m = 1$ and (V, E) is a tree with a degree bounded by two. If $n = 0$ then T is a many-sorted term and if $n = 1$ then T is a many-sorted term with an unknown. By Term_σ we denote the set of terms over the signature[3] σ and by $\mathsf{Term}_\sigma[X]$ we denote the set of terms with an unknown over σ.

[2] This is no restriction as an n-ary operations can be simulated by a small set of binary operations.

[3] Note that for a term to be over some signature, the signature has to admit 0-ary operations since we need them for the leafs.

We assume that the term (with or without an unknown) is encoded as a string over a fixed alphabet[4]. We call this a linearization of a term. The details of this are presented in [22]. From now on we will not distinguish between terms and their linearizations.

2.3 Semantics: evaluation of circuits and terms without unknown

So far we have defined circuits and terms (with and without unknowns) syntactically. Now we will give semantics for circuits and terms. Informally, the semantics of a circuit is the tuple of functions its outputs compute and that of a term is the value which its output evaluates to. We make this notion formal by introducing the notion of a many-sorted algebra, which helps us assign semantics to the operations in the circuits/terms.

▶ **Definition 4** (Many-sorted universal algebra)**.** Given a many-sorted signature σ with S sorts, a many-sorted universal algebra is a tuple $\mathcal{A} = (\{\mathbb{D}_1, \ldots, \mathbb{D}_S\}, \circledast_1, \ldots, \circledast_{|\sigma|})$ where $\circledast_i \colon \mathbb{D}_{\mathsf{In}_\sigma(i,1)} \times \ldots \times \mathbb{D}_{\mathsf{In}_\sigma(i,\alpha_i)} \to \mathbb{D}_{\mathsf{Out}_\sigma(i)}$ where $\alpha_i = |\mathsf{In}_\sigma(i)|$. We call the sets $\mathbb{D}_i$ subdomains and the union of all subdomains $\mathbb{D}$, which is the domain.

From now on we will simply say algebra instead of many-sorted universal algebra. Given an algebra which has the same signature as a circuit and valuations for the input gates, we can evaluate the circuit under the given algebra. Note that this in turn can be used to evaluate terms since terms are just circuits that are trees without inputs.

▶ **Definition 5** (Evaluation of many-sorted circuits)**.** Given an algebra $\mathcal{A}$ over signature σ and a word $w \in \mathbb{D}^n$, the evaluation map $\eta_{\mathcal{A},w} \colon \mathsf{Circ}_{\sigma,n,m} \to \mathbb{D}^m$ is a map defined inductively for all $v \in V$. Here, let $T(v)$ be the maximal subcircuit of a circuit C containing all nodes from which v is reachable.

- If $\mathsf{Gatetype}(v) = x_i$ then $\eta_{\mathcal{A},w}(T(v)) = w_i$ provided $w_i \in \mathbb{D}_j$ implies that $\mathsf{In}_C(i) = j$.
- Let α be the arity $|\mathsf{In}_\sigma(\mathsf{Gatetype}(v))|$ and $v_1, \ldots v_k$ be the predecessors of v ordered by their output wire order, then $\eta_{A,w}(T(v)) = \circledast_{\mathsf{Gatetype}(v)}(\eta_{\mathcal{A},w}(T(v_1)), \ldots, \eta_{\mathcal{A},w}(T(v_\alpha)))$.

Let $v_1, \ldots v_m$ be the output gates and C a circuit, then $\eta_{\mathcal{A},w}(C) = (\eta_{\mathcal{A},w}(T(v_1)), \ldots, \eta_{\mathcal{A},w}(T(v_m)))$ if for all $\mathsf{Outputgates}^{-1}(y_i) = v_i$ the following holds: $\mathsf{Out}_C(i) = \mathsf{Out}_\sigma(\mathsf{Gatetype}(v_i))$.

2.4 Semantics: evaluation of terms with an unknown

We have now described how to evaluate terms and circuits. However, we would like to treat terms with an unknown slightly differently. We do not give a value to the unknown but we let this term evaluate to a function. If some algebra is given, the set of functions we can get can be obtained from this algebra. In fact we now get a many-sorted algebra since it needs to contain the original algebra as well as these functions. There are operations of mixed sorts. We can e.g. combine a function $\mathbb{D} \to \mathbb{D}$ and a value $\mathbb{D}$.

▶ **Definition 6** (Functional algebra)**.** Given an algebra $\mathcal{A} = (\mathbb{D}, \circledast_1, \ldots, \circledast_k)$ over a single-sorted signature σ which only contains operations that are at most binary, the functional algebra is defined to be $\mathcal{F}(\mathcal{A}) = (\{\mathbb{D}, \widetilde{\mathbb{D}}\}, F)$, where F is a placeholder for the operations which we will define next and $\widetilde{\mathbb{D}} \subseteq \mathbb{D}^{\mathbb{D}}$ is the smallest set containing the identity function and is closed under the operations in F which are the following:

- All operations of $\mathcal{A}$: $\circledast_1, \ldots, \circledast_k$.

[4] It is the usual way of encoding machines as strings.

- $\circ\colon \widetilde{\mathbb{D}} \to \widetilde{\mathbb{D}}$ which is the functional composition.
- An operation for functional evaluation $\odot\colon \widetilde{\mathbb{D}} \times \mathbb{D} \to \mathbb{D}$, where $f \odot c = f(c)$.
- For each $\circledast_i\colon \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ there are two variants: $\overleftarrow{\circledast}_i\colon \widetilde{\mathbb{D}} \times \mathbb{D} \to \widetilde{\mathbb{D}}$ and $\overrightarrow{\circledast}\colon \mathbb{D} \times \widetilde{\mathbb{D}} \to \widetilde{\mathbb{D}}$, where $(f \overleftarrow{\circledast}_i c)(x) = f(x) \circledast_i c$ and $(c \overrightarrow{\circledast}_i f)(x) = c \circledast_i f(x)$.
- For each $\circledast_i\colon \mathbb{D} \to \mathbb{D}$ there is $\widetilde{\circledast}_i\colon \widetilde{\mathbb{D}} \to \widetilde{\mathbb{D}}$, where $(\widetilde{\circledast}_i f)(x) = \circledast_i f(x)$.

The signature of $\mathcal{F}(\mathcal{A})$ is denoted by $\sigma(\mathcal{F}(\mathcal{A}))$.

This definition can be lifted to arbitrary arities. The evaluation of terms with an unknown can now be done using the previous definition. Again we use the linearization.

▶ **Definition 7** (Evaluation of terms with an unknown)**.** Let $\mathcal{A} = (\mathbb{D}, \circledast_1, \dots, \circledast_k)$ be an algebra over a single-sorted signature σ with maximal operation arity of two and let $\mathcal{F}(\mathcal{A})$ be the functional algebra of $\mathcal{A}$. The evaluation map $\mu_{\mathcal{A}}\colon \mathsf{Term}_\sigma[X] \to \widetilde{\mathbb{D}}$ is a map defined inductively for all $i \in [k]$:

- $\mu_{\mathcal{A}}(X) = \mathrm{id} \in \widetilde{\mathbb{D}}$,
- $\mu_{\mathcal{A}}(\oplus_i f) = \widetilde{\circledast}_i \mu_{\mathcal{A}}(f)$ for $f \in \mathsf{Term}_\sigma[X]$,
- $\mu_{\mathcal{A}}(f \oplus_i t) = \mu_{\mathcal{A}}(f) \overleftarrow{\circledast}_i \eta_{\mathcal{A}}(t)$ where $f \in \mathsf{Term}_\sigma[X]$ and $t \in \mathsf{Term}_\sigma$,
- $\mu_{\mathcal{A}}(t \oplus_i f) = \eta_{\mathcal{A}}(t) \overrightarrow{\circledast}_i \mu_{\mathcal{A}}(f)$ where $f \in \mathsf{Term}_\sigma[X]$ and $t \in \mathsf{Term}_\sigma$.

2.5 Interpreting classical circuit classes in this framework

In order to view the classical circuit complexity classes in this framework, we will first start with the definitions of family of algebras and family of many-sorted circuits.

▶ **Definition 8** (Family of algebras)**.** A family of algebras $(\mathcal{A}_n)_{n\in\mathbb{N}}$ is a sequence of algebras, where $\mathcal{A}_i = (\{\mathbb{D}_1^{p_1(i)}, \dots, \mathbb{D}_S^{p_S(i)}\}, \circledast_1^i, \dots, \circledast_k^i)$ for $i \in \mathbb{N}$ and p_j being polynomials for $j \in [S]$[5].

Here, let us assume that the circuits have one output gate and all input gates have the same sort[6]. Given an algebra $\mathcal{A}$ let $\mathbb{D}_I$ and $\mathbb{D}_O$ be the two subdomains that correspond to the input and output values, respectively. Then a circuit C_n of n inputs realizes a function $F_{\mathcal{A}}(C_n)\colon \mathbb{D}_I^n \to \mathbb{D}_O$. Given a family of circuits C we then get a function $F_{\mathcal{A}}(C) = \mathbb{D}_I^* \to \mathbb{D}_O$.

In general, assuming a family of circuits is very powerful, so in complexity it is natural to require that this family is computable in some complexity bound. We then speak of uniformity. Our constructions will be DLOGTIME-uniform. (See e.g. [34] or [35] for basics in circuit complexity.) We use our framework to define the classical Boolean version of $\mathbf{NC}^i$. Here $\mathcal{B}$ is the Boolean algebra $(\mathbb{B}, \wedge, \vee, \neg, \bot, \top)$.

▶ **Definition 9** ($\mathbf{NC}^i$)**.** The set $\mathbf{NC}^i$ contains all functions $F_{\mathcal{B}}(C)$, where C is a family of circuits of signature same as $\mathcal{B}$ that contains circuits of polynomial size and $\mathcal{O}(\log^i n)$ depth.

Note that the bounded fan-in of the gates is ensured through the signature of $\mathcal{B}$. For defining $\mathbf{AC}^i$ we need a different algebra to handle the unbounded fan-in gates. In fact we need a family of algebras $\mathcal{B}^* = (\mathcal{B}_n)_{n\in\mathbb{N}}$ where $\mathcal{B}_i = (\mathbb{B}, \wedge, \vee, \neg, \bot, \top, \wedge_i, \vee_i)$, $\wedge_i\colon \mathbb{B}^i \to \mathbb{B}$ and $\vee_i\colon \mathbb{B}^i \to \mathbb{B}$. Hence this is an example where the difference between the members of the families only lies in the arity of operations.

▶ **Definition 10** ($\mathbf{AC}^i$)**.** The set $\mathbf{AC}^i$ contains all functions $F_{(\mathcal{B}_n)_{n\in\mathbb{N}}}(C)$, where $C = (C_n)_{n\in\mathbb{N}}$ is a family of circuits that contains circuits of polynomial size, $\mathcal{O}(\log^i n)$ depth and where C_n has the same signature as $\mathcal{B}_n$.

[5] Note that we are assuming a family of signatures here, rather than a single signature.
[6] This is usually the case in the classical circuit complexity models and hence the assumption.

The classes $\mathbf{SAC}^i$ we also get in a similar way as $\mathbf{AC}^i$; we only have to replace the unbounded AND gates $\wedge_i$ from the algebras with the bounded fan-in AND gates.

2.6 Composition of algebras and extended circuit classes

We will now define circuit classes which are variants of the previously defined classes obtained using some algebra $\mathcal{A}$. These are circuits that have Boolean gates as well as $\mathcal{A}$-gates. In our context, such circuits arise in the algorithm design stage. Therefore, we ensure by design that in these circuits Boolean values and values of $\mathcal{A}$ interact via multiplexer gates defined below.

▶ **Definition 11** (Multiplexer operation)**.** Given a domain $\mathbb{D}$, the ternary multiplexer operation is defined as $\mathrm{mp}_{\mathbb{D}} \colon \mathbb{B} \times \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ with

$$(b, d_0, d_1) \mapsto \begin{cases} d_0 & \text{if } b = 0 \\ d_1 & \text{else} \end{cases} .$$

Now we can use multiplexer operations to compose algebras which have as subalgebras the Boolean one $\mathcal{B}$ and some other algebra $\mathcal{A}$ and they interact via multiplexer operations.

▶ **Definition 12** (Composition of algebras)**.** Given an algebra $\mathcal{A} = (\{\mathbb{D}_1, \ldots, \mathbb{D}_S\}, \circledast_1, \ldots, \circledast_k)$, by $(\mathcal{B}, \mathcal{A})$ we denote the algebra $(\{\mathbb{B}, \mathbb{D}_1, \ldots, \mathbb{D}_S\}, \wedge, \vee, \neg, \circledast_1, \ldots, \circledast_k, (\mathrm{mp}_{\mathbb{D}_i})_{i\in[S]})$. Given an algebra of the form $(\mathcal{B}, \mathcal{A})$ and an algebra $\mathcal{A}' = (\{\mathbb{D}'_1, \ldots, \mathbb{D}'_{|S'|}\}, \circledast'_1, \ldots, \circledast'_{k'})$, by $((\mathcal{B}, \mathcal{A}), \mathcal{A}')$ we denote the algebra

$$(\{\mathbb{B}, \mathbb{D}_1, \ldots, \mathbb{D}_S, \mathbb{D}'_1, \ldots, \mathbb{D}'_{|S'|}\}, \wedge, \vee, \neg, \circledast_1, \ldots, \circledast_k, \circledast'_1, \ldots, \circledast'_{k'}, (\mathrm{mp}_{\mathbb{D}_i})_{i\in[S]}, (\mathrm{mp}_{\mathbb{D}'_i})_{i\in[S']}).$$

Hence we write $(\mathcal{B}, \mathcal{A}_1, \ldots, \mathcal{A}_k) = ((\mathcal{B}, \mathcal{A}_1, \ldots, \mathcal{A}_{k-1}), \mathcal{A}_k)$ for the composition of k algebras.

Note that the previous definition also naturally carries over to families of algebras. We can define classes similar to e.g. $\mathbf{NC}^i$ that are enriched by some algebra. Intuitively, the Boolean part is directing the non-Boolean part via multiplexer gates.

▶ **Definition 13** ($\mathcal{A}$-$\mathbf{NC}^i$, $\mathcal{A}$-$\mathbf{NC}^i_{\mathbb{D}}$)**.** The set $\mathcal{A}$-$\mathbf{NC}^i_{\mathbb{D}}$ contains all functions $F_{(\mathcal{B},\mathcal{A})}(C)$, where C is a family of circuits having the same family of signatures as $(\mathcal{B}, \mathcal{A})$ that contains circuits of polynomial size, depth $\log^i n$, inputs of $\mathbb{D}$ and one output of a subdomain of $\mathcal{A}$. For the special case of Boolean inputs we set $\mathcal{A}$-$\mathbf{NC}^i = \mathcal{A}$-$\mathbf{NC}^i_{\mathbb{B}}$.

The class $(\mathbb{N}, +, \times, 0, 1)$-$\mathbf{NC}^1$ is in fact $\#\mathbf{NC}^1$ and $(\mathbb{Z}, +, \times, 0, 1)$-$\mathbf{NC}^1$ is the well-studied Gap$\mathbf{NC}^1$. The $\mathcal{A}$ -$\mathbf{NC}^i$ and $\mathcal{A}$-$\mathbf{NC}^i_{\mathbb{D}}$ definitions naturally carry over to classes other than $\mathbf{NC}^i$. The idea of $\mathcal{A}$-$\mathbf{NC}^i_{\mathbb{D}}$ is that we allow Boolean and non-Boolean inputs which then help in composing such circuits, i.e. the output of one circuit is the input of another one.

3 Term evaluation algorithm

Given some term and an algebra $\mathcal{A}$ of the same signature, what does the term evaluate to over $\mathcal{A}$? This problem is the term evaluation problem. The purpose of this section is to prove our main theorem regarding the parallel algorithm for the term evaluation problem.

▶ **Theorem 14** (Main Theorem)**.** *Given an algebra $\mathcal{A}$ of single-sorted signature σ and domain $\mathbb{D}$, the evaluation function $\eta_{\mathcal{A}} \colon \mathsf{Term}_\sigma \to \mathbb{D}$ can be computed in $\mathcal{F}(\mathcal{A})$-$\mathbf{NC}^1$. Moreover, the construction ensures that the we get a DLOGTIME-uniform $\mathcal{F}(\mathcal{A})$-$\mathbf{NC}^1$ circuit family.*

Note that the theorem and its proof are independent of the actual algebra $\mathcal{A}$. The algebra can be arbitrary, with no restrictions such as commutativity or associativity on the operations.

The rest of the section is devoted to proving the above theorem. Many details are omitted in the this version due to shortage of space. (See [22] for those details.)

3.1 Representing terms in a normal form

Recall that we assume without loss of generality that all the operations used in the terms are binary. A term $(\phi \circledast \psi)$, where ϕ and ψ are also terms, is in infix notation. If ϕ' and ψ' are the equivalent postfix notations for terms ϕ and ψ, then $\phi'\psi'\circledast$ is the postfix equivalent of $(\phi \circledast \psi)$. Note that parentheses are not needed in the postfix notation.

A postfix term is in *postfix normal form (PNF)* if for subterms ϕ and ψ of $\phi\psi\circledast$, $|\phi| \geq |\psi|$. This normal-form was defined by Buss to design an **NC** algorithm [5] for evaluating Boolean formulas. Our algorithm has its roots in this and other related works [7, 15]. We are aware of a simplified version [6] of [5] which directly operates on the infix notation, however we use PNF as that is more convenient here. The details of how the given term is converted into the PNF form can be found in [22]. A crucial property of a term in PNF is that it always has suffixes that are *open* terms, which correspond to terms with an unknown.

3.2 Dividing terms: some structural lemmas regarding terms

Let T be a term given as a string of length n in PNF. Let $[l, r]$ be a subinterval. The size of the interval is $s = r - l + 1$. Typically, an interval is specified by l and r and additionally a position m which is approximately the middle position. It is either can be $\lfloor l + \frac{s}{2} \rfloor$ or $\lceil l + \frac{s}{2} \rceil$. Its exact position is not decided a priori, but comes through a recursive evaluation of subintervals of $[l, r]$. We also use *interval borders* $l' = \lfloor l + \frac{s}{3} \rfloor$ and $r' = \lceil l + \frac{2s}{3} \rceil$. This divides the interval $[l, r]$ in approximately thirds. We not only consider the three intervals but also the intervals obtained by concatenating first two thirds and the second two thirds. These five intervals will be our recursion intervals. Based on those static intervals we define some dynamic intervals, i.e. intervals depending on the specific input:

- The largest closed or open subterm in $[l, r]$ that contains m. This interval is denoted as $\mathcal{M}(l, m, r) = [\mathcal{M}^1(l, m, r), \mathcal{M}^2(l, m, r)]$.
- The open subterm in $[l, r]$ that begins with $\max\{p \mid l \leq p \leq m \wedge l-1 <_T p-1 \wedge l-1 \not<_T p\}$ and ends with the largest position $q \in [m, r]$ such that $[p, q]$ is an open subterm. This interval is denoted as $\mathcal{N}(l, m, r) = [\mathcal{N}^1(l, m, r), \mathcal{N}^2(l, m, r)]$.
- The largest open subterm in $[l, r]$ that precedes $\mathcal{M}(l, m, r)$. This interval is denoted as $\mathcal{L}(l, m, r) = [\mathcal{L}^1(l, m, r), \mathcal{L}^2(l, m, r)]$. It is $\mathcal{L}^2(l, m, r) + 1 = \mathcal{M}^1(l, m, r)$.
- The largest open subterm in $[l, r]$ that follows $[\mathcal{M}^1(l, m, r), \mathcal{M}^2(l, m, r) + 1]$. This interval is denoted as $\mathcal{R}(l, m, r) = [\mathcal{R}^1(l, m, r), \mathcal{R}^2(l, m, r)]$. It is $\mathcal{R}^1(l, m, r) - 2 = \mathcal{M}^2(l, m, r)$ and $\mathcal{M}^2(l, m, r) + 1$ is a binary operation symbol. If this operation exists, we denote the set containing its position by $O(l, m, r) = \{\mathcal{M}^2(l, m, r) + 1\}$.

If values of (l, m, r) are clear from the context, we drop them. Note that an $\mathcal{M}$ interval could correspond to an open or a closed term. The $\mathcal{N}, R$ intervals always correspond, by definition, to open terms. (That is the only way we use them in the algorithm.) The $\mathcal{L}$ interval is also defined to be open however even if we allowed it to be closed, it can only be an open term. This is because it is strictly shorter than $\mathcal{M} \cup \mathcal{N}$ and does not contain the complete second operand of O. Figure 1 shows the considered intervals. The intervals might

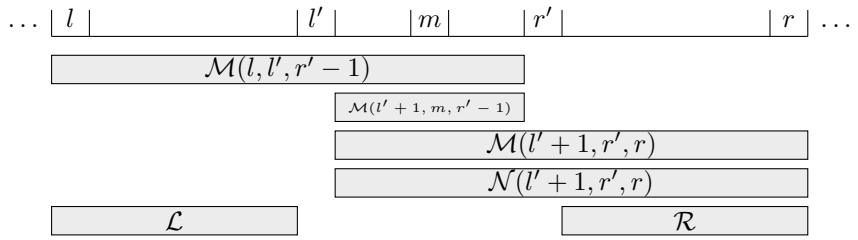


Figure 1 The figure shows how a recursion interval is subdivided into smaller recursion intervals. In this case the subdivision for computing $\mathcal{M}(l, m, r)$ is shown. The six intervals yield recursively six values which may be used to be combined in order to get the evaluation of the $\mathcal{M}(l, m, r)$ interval.

be empty. Note that the intervals are well-defined. We now state some properties (see proofs in the full version [22]) regarding these intervals.

▶ **Lemma 15.** *Given intervals $[p_1, q_1] \subseteq [l, r]$ and $[p_2, q_2] \subseteq [l, r]$ which address closed or open terms with $[p_1, q_1] \cap [p_2, q_2] \neq \emptyset$ then $[p_1, q_1] \cup [p_2, q_2]$ is also a closed or open term.*

▶ **Lemma 16.** *It holds $\mathcal{M}^2 = \mathcal{N}^2$ and $\mathcal{M}^2(l, l', r' - 1) + 1 = \mathcal{N}^1(l' + 1, r', r)$.*

The key lemmas which later constitute the recursive evaluation algorithm are given below. They show how the algorithm actually composes a term by subterms coming from static subintervals. For instance, Figure 1 shows the subintervals relevant for the next lemma.

▶ **Lemma 17.** *Given a term T and subinterval $[l, r]$ with middle m, $\mathcal{M}$ equals one of the following intervals: 1. $\mathcal{M}(l, l', r' - 1)$, 2. $\mathcal{M}(l' + 1, r', r)$, 3. $\mathcal{M}(l' + 1, m, r' - 1)$,*
4. $\mathcal{M}(l, l', r' - 1) \cup \mathcal{N}(l' + 1, r', r)$,
5. $\mathcal{L}(l, m, r) \cup \mathcal{M}(l, l', r' - 1) \cup \mathcal{N}(l' + 1, r', r) \cup O(l, m, r) \cup \mathcal{R}(l, m, r)$. Further the sets involved in the unions of case 4 and 5 are disjoint unions.

In a very similar way we can treat $\mathcal{N}$ as well.

▶ **Lemma 18.** *Given a term T and subinterval $[l, r]$ with middle m, $\mathcal{N}$ equals one of the following intervals: 1. $\mathcal{N}(l, l', r' - 1)$, 2. $\mathcal{N}(l' + 1, r', r)$, 3. $\mathcal{N}(l' + 1, m, r' - 1)$,*
4. $\mathcal{N}(l, l', r' - 1) \cup \mathcal{N}(l' + 1, r', r)$,
5. $\mathcal{N}(l, l', r' - 1) \cup \mathcal{N}(l' + 1, r', r) \cup O(l, m, r) \cup \mathcal{R}(l, m, r)$. Further the sets involved in the unions of case 4 and 5 are disjoint.

The intervals $\mathcal{M}$ and $\mathcal{N}$ are built around the property of containing a middle position m. The intervals $\mathcal{L}$ and $\mathcal{R}$ are different. They can lie arbitrarily in $[l, l' - 1]$ resp. $[r' + 1, r]$ and we initially do not know anything about the location of the middle points. Our goal is to reduce $\mathcal{L}$ and $\mathcal{R}$ to some $\mathcal{M}(\bar{l}, \bar{m}, \bar{r})$ where find $\bar{l}$, $\bar{m}$, and $\bar{r}$ using a binary search.

▶ **Lemma 19.** *Given a term T and subinterval $[l, r]$ with middle m, there is an interval $[\bar{l}, \bar{r}] \subseteq [l, l' - 1]$ with middle $\bar{m}$ such that $\mathcal{L} = \mathcal{M}(\bar{l}, \bar{m}, \bar{r})$ and this $\bar{m}$ can be computed by a binary search inside an appropriate range defined by l, m, r.*

▶ **Lemma 20.** *Given a term T and subinterval $[l, r]$ with middle m there is an interval $[\bar{l}, \bar{r}] \subseteq [r' + 1, r]$ with middle $\bar{m}$ such that $\mathcal{R} = \mathcal{M}(\bar{l}, \bar{m}, \bar{r})$ and $\bar{m}$ can be computed by a binary search inside an appropriate range defined by l, m, r.*

3.3 The evaluation algorithm

The algorithm we present is a recursive one which is given in terms of circuits. Lemmas 17, 18, 19, and 20 directly suggest how the recursive evaluation will work: to evaluate an interval compute smaller fixed subintervals and use their evaluations to obtain the overall evaluation.

In particular, we proceed as follows: we wish to compute $\mathcal{M}(1, \lfloor n/2 \rfloor, n)$. In order to do that, we design a procedure called Eval, which at any given stage of recursion receives as input an $\mathcal{M}, \mathcal{N}, \mathcal{L}$ or an $\mathcal{R}$ interval along with the current values of l, m, r.

If it is an $\mathcal{M}$ or an $\mathcal{N}$ interval then the procedure first determines which among the five cases applies for the subsequent recursion call. By cases, we mean the five possibilities listed in Lemmas 17, 18. Moreover, if it is an $\mathcal{M}$ interval then it also determines whether the term defined by it is an open term or a closed term and keeps that information in a local flag. (Say the flag is by default set to 0 and then it is toggled to 1 if the term is closed.)

If it is an $\mathcal{L}$ or an $\mathcal{R}$ interval then it first computes the appropriate $\bar{l}$, $\bar{r}$ and $\bar{m}$ values and makes recursive calls for the appropriate $\mathcal{M}$ interval defined using these $\bar{l}, \bar{r}, \bar{m}$ values.

Finally, once the recurive calls return the values, the Eval procedure combines these values. If the flag is set to 1 then we know that the current call is dealing with a closed term and therefore, the procedure outputs an evaluated value. On the other hand, if the flag is false, the procedure outputs a function from $\mathbb{D} \to \mathbb{D}$.

For the evaluation we need to implement circuits which on a given interval $[l, r]$ evaluate the intervals $\mathcal{M}$, $\mathcal{N}$, $\mathcal{L}$, and $\mathcal{R}$. In the case of $\mathcal{M}$ we need to distinguish whether the evaluation is a value of $\mathbb{D}$ or a function of $\widetilde{\mathbb{D}}$. In the other cases the result is always a function. Correspondingly, the following circuits may arise: Eval$(\mathcal{M}(l, m, r))$, Eval$(\mathcal{N}(l, m, r))$, Eval$(\mathcal{L}(l, m, r), \bar{l}, \bar{m}, \bar{r})$, Eval$(\mathcal{R}(l, m, r), \bar{l}, \bar{m}, \bar{r})$.

The variables $\bar{l}, \bar{m}, \bar{r}$ exist to serve the binary search as mentioned in Lemma 19 and 20.

Those circuits all work in a similar way: Depending on the structure of the term one of a number of cases holds which determines how the output value is composed of the recursion results. So the recursion results are combined according to the cases and then fed into a multiplexer-gate which chooses the correct output. The circuits determining the cases are called Case$(\mathcal{M}(l, m, r))$, Case$(\mathcal{N}(l, m, r))$, Case$(\mathcal{L}(l, m, r), \bar{l}, \bar{m}, \bar{r})$, Case$(\mathcal{R}(l, m, r), \bar{l}, \bar{m}, \bar{r})$.

Here we have already assumed that the term is given in the PNF form. To ensure this, as the first step we perform this conversion to the PNF form as done in [5]. The conversion is of complexity $\mathbf{TC}^0$. The resulting term is T as above. The details of the working of various recursive calls and the claimed complexity bounds are presented in the full version [22].

4 Application

Our main theorem can be used for many different applications. We first present a template or recipe used for deriving these applications. It consists of the following steps:

1. **Find an algebra $\mathcal{A}$.** Given a problem Π, which could be a language or a function, find an algebra $\mathcal{A}_\Pi = (\mathbb{D}, \circledast_1, \ldots, \circledast_k)$, such that Π reduces to term evaluation over $\mathcal{A}_\Pi$.
2. **Find a coding for $\mathcal{F}(\mathcal{A}_\Pi)$.** Now we know by our main theorem that Π is in $\mathcal{F}(\mathcal{A}_\Pi)$-$\mathbf{NC}^1$. However what we want is a "real" class like $\mathbf{NC}^1$ or $\#\mathbf{SAC}^7$. Hence we encode $\mathcal{F}(\mathcal{A}_\Pi)$ in a way that we end up with a Boolean or an arithmetic class. So find a code c mapping (details in [22]) into a family of algebras, that have domains based on $\mathbb{B}$, $\mathbb{N}$, or $\mathbb{Z}$, depending on whether we wish to prove Boolean or arithmetic circuit upper bounds.
3. **Analyze the complexity of the operations used in $c(\mathcal{F}(\mathcal{A}_\Pi))$-$\mathbf{NC}^1$.** Now we know that Π is in $c(\mathcal{F}(\mathcal{A}_\Pi))$-$\mathbf{NC}^1$ since the coding admits a reduction. If we have chosen c

well, we can implement the operations of $c(\mathcal{F}(\mathcal{A}_\Pi))$ efficiently. Note that $c(\mathcal{F}(\mathcal{A}_\Pi))$ could be a family. The members of the family all contain the following coded operations:

- The operations of $\mathcal{A}_\Pi$: $\circledast_1^c \dots \circledast_k^c$.
- The functional versions for each binary operation $\circledast_i$: $\overleftarrow{\circledast}_i^c$ and $\overrightarrow{\circledast}_i^c$. Recall that $\overleftarrow{\circledast}_i : \mathbb{D}^\mathbb{D} \times \mathbb{D} \to \mathbb{D}^\mathbb{D}$ and $\overrightarrow{\circledast}_i : \mathbb{D} \times \mathbb{D}^\mathbb{D} \to \mathbb{D}^\mathbb{D}$.
- The functional versions for each unary operation $\circledast_i$ which is $\widetilde{\circledast}_i^c : \mathbb{D}^\mathbb{D} \to \mathbb{D}^\mathbb{D}$.
- The functional composition of $\mathcal{F}(\mathcal{A}_\Pi)$: $\circ^c : \mathbb{D}^\mathbb{D} \times \mathbb{D}^\mathbb{D} \to \mathbb{D}^\mathbb{D}$.
- The function application operation of $\mathcal{F}(\mathcal{A}_\Pi)$: $\odot^c : \mathbb{D}^\mathbb{D} \times \mathbb{D} \to \mathbb{D}$.

An algebra usually also has 0-ary operations, but here there is no complexity to analyze. The following is not a part of the algebra but comes into play in the construction of the $c(\mathcal{F}(\mathcal{A}_\Pi))$-$\mathbf{NC}^1$ circuits: Multiplexer operations for all subdomains $\mathbb{D}$ of $c(\mathcal{F}(\mathcal{A}_\Pi))$: $\mathrm{mp}_\mathbb{D}$. These operations are used in the $c(\mathcal{F}(\mathcal{A}_\Pi))$-$\mathbf{NC}^1$ circuit as black boxes. In this third step we have to come up with a efficient implementation of all these operations in order to derive a good upper bound. In general, the depth increases by a logarithmic factor when comparing the complexity of the functions and the overall circuit. So, if, say, all the functions are in $\#\mathbf{NC}^1_\mathbb{D}$, then $c(\mathcal{F}(\mathcal{A}_\Pi))$-$\mathbf{NC}^1 \subseteq \#\mathbf{NC}^2$.

All the applications we show follow this template. A simple application regarding arithmetic formula evaluation is presented here. Due to lack of space, the rest of the applications are omitted from this version. (See [22] for the other applications.)

4.1 Evaluating arithmetic terms and distributive algebras

We consider evaluating terms over $\mathcal{N} = (\mathbb{N}, +, \times, 0, 1)$ and $\mathcal{Z} = (\mathbb{Z}, +, \times, 0, 1)$.

▶ **Theorem 21** ([7]). *Evaluating terms over $\mathcal{N}$ ($\mathcal{Z}$) is in $\#\mathbf{NC}^1$ (Gap$\mathbf{NC}^1$ resp.).*

Proof. We give the proof for $\mathcal{N}$. The case of $\mathcal{Z}$ is handled similarly.

step 1. The problem is directly a term evaluation problem, hence no reduction is needed.

step 2. Consider the algebra $\mathcal{F}(\mathcal{N}) = (\{\mathbb{N}, \widetilde{\mathbb{N}}\}, +, \times, 0, 1, \overleftarrow{+}, \overleftarrow{\times}, \overrightarrow{+}, \overrightarrow{\times}, \circ, \odot)$, where $\widetilde{\mathbb{N}} \subseteq \mathbb{N}^\mathbb{N}$. We choose a coding c such that $c(\mathbb{N}) = \mathbb{N}$ and $c(\widetilde{\mathbb{N}}) = \mathbb{N}^2$. The functions in $\widetilde{\mathbb{N}}$ are of the form $x \mapsto ax + b$ for some $a, b \in \mathbb{N}$: We begin with the identity function $x \mapsto 1x + 0$ which is clearly of this form. Now we show that the operations of $\mathcal{F}(\mathcal{N})$ keep functions in this form.

- $\circ^c$: Given some functions $f(x) = a_f x + b_f$ and $g(x) = a_g x + b_g$, then $f \circ g$ is of this form: $x \mapsto a_f a_g x + a_f b_g + b_f$. So $c(f \circ g) = c(f) \circ^c c(g) = (a_f, b_f) \circ^c (a_g, b_g) = (a_f a_g, a_f b_g + b_f)$.
- $\overleftarrow{+}^c$: Consider $c(f + e)$ for $f \in \mathbb{N}^\mathbb{N}$ and $e \in \mathbb{N}$. Now $c(f) +^c c(e) = (a, b) +^c e = (a, b + e)$ where $f(x) = ax + b$. The operation $\overrightarrow{+}^c$ follows similarly.
- $\overleftarrow{\times}^c$: Consider $c(f \times e)$ for $f \in \mathbb{N}^\mathbb{N}$ and $e \in \mathbb{N}$. Now $c(f) +^c c(e) = (a, b) +^c e = (a \times e, b \times e)$ where $f(x) = ax + b$. The operation $\overrightarrow{\times}^c$ follows similarly.

This shows that c is indeed a valid coding.

step 3. We now have an upper bound of $c(\mathcal{F}(\mathcal{B}))$-$\mathbf{NC}^1$. As all operations use constantly many inputs of natural numbers, there exist arithmetic circuit implementations for all operations. Further, all Boolean gates and multiplexer gates can be simulated by arithmetic circuit constructions, so all operations are in $\#\mathbf{NC}^0_\mathbb{N}$. Hence we get $c(\mathcal{F}(\mathcal{B}))$-$\mathbf{NC}^1 \subseteq \#\mathbf{NC}^1$. ◀

In the previous proof we used distributivity of $+$ and $\times$ which allows us to represent functions by two values. This we can do in general, so we get the following:

▶ **Theorem 22.** *Given a distributive algebra* $\mathcal{A} = (\mathbb{D}, \circledast_1, \circledast_2)$*, then evaluating terms over* $\mathcal{A}$ *is in* $\mathcal{A}$-$\mathbf{NC}^1$.

5 Discussion

We have seen that many problems that have a tree-like structure can be solved in parallel using our term evaluation algorithm. The list of problem we covered here should indicate the potential of the framework. We expect that there will be many more applications, which can be derived using the unifying framework presented here.

References

1 Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in nc^ 2. In *STACS*, volume 4393, pages 489–499, 2007.

2 Eric Allender and Ian Mertz. Complexity of regular functions. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2015. doi:10.1007/978-3-319-15579-1_35.

3 Nikhil Balaji, Samir Datta, and Venkatesh Ganesan. Counting euler tours in undirected bounded treewidth graphs. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 246–260. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.246.

4 Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974. doi:10.1145/321812.321815.

5 Samuel R. Buss. The boolean formula value problem is in ALOGTIME. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 123–131. ACM, 1987. doi:10.1145/28395.28409.

6 Samuel R. Buss. Algorithms for boolean formula evaluation and for tree contraction. In *Arithmetic, Proof Theory and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

7 Samuel R. Buss, Stephen A. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992. doi:10.1137/0221046.

8 Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985. doi:10.1016/S0019-9958(85)80041-3.

9 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

10 Patrick W. Dymond. Input-driven languages are in log n depth. *Inf. Process. Lett.*, 26(5):247–250, 1988. doi:10.1016/0020-0190(88)90148-2.

11 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

12 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Guest column: Parallel algorithms for perfect matching. *SIGACT News*, 48(1):102–109, 2017. doi:10.1145/3061640.3061655.

13 Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Theory of Computing Systems*, 17(1):13–27, 1984.

14 Moses Ganardi and Markus Lohrey. A universal tree balancing theorem. *CoRR*, abs/1704.08705, 2017. arXiv:1704.08705.

15 A. Gupta. A fast parallel algorithm for recognition of parenthesis languages. Master's thesis, University of Toronto, 1985.

16 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM J. Comput.*, 45(3):1064–1079, 2016. doi:10.1137/140957123.

17 Johan Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

18 Johan Håstad. On the correlation of parity and small-depth circuits. *SIAM J. Comput.*, 43(5):1699–1708, 2014. doi:10.1137/120897432.

19 Joseph JáJá. *An introduction to parallel algorithms*, volume 17. Addison-Wesley Reading, 1992.

20 Maurice J. Jansen and Jayalal Sarma. Balancing bounded treewidth circuits. *Theory Comput. Syst.*, 54(2):318–336, 2014. doi:10.1007/s00224-013-9519-3.

21 Andreas Krebs, Nutan Limaye, and Michael Ludwig. Cost register automata for nested words. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, volume 9797 of *Lecture Notes in Computer Science*, pages 587–598. Springer, 2016. doi:10.1007/978-3-319-42634-1_47.

22 Andreas Krebs, Nutan Limaye, and Michael Ludwig. A unified method for placing problems in polylogarithmic depth. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:19, 2017. URL: https://eccc.weizmann.ac.il/report/2017/019.

23 Andreas Krebs, Nutan Limaye, and Meena Mahajan. Counting paths in VPA is complete for $\#\mathrm{NC}^1$. In My T. Thai and Sartaj Sahni, editors, *Computing and Combinatorics, 16th Annual International Conference, COCOON 2010, Nha Trang, Vietnam, July 19-21, 2010. Proceedings*, volume 6196 of *Lecture Notes in Computer Science*, pages 44–53. Springer, 2010. doi:10.1007/978-3-642-14031-0_7.

24 Andreas Krebs, Nutan Limaye, and Meena Mahajan. Counting paths in VPA is complete for #nc 1. *Algorithmica*, 64(2):279–294, 2012. doi:10.1007/s00453-011-9501-x.

25 Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms*, volume 400. Benjamin/Cummings Redwood City, 1994.

26 Markus Lohrey. On the parallel complexity of tree automata. In Aart Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2001. doi:10.1007/3-540-45127-7_16.

27 Nancy A. Lynch. Log space recognition and translation of parenthesis languages. *J. ACM*, 24(4):583–590, 1977. doi:10.1145/322033.322037.

28 Meena Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In Michael E. Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana.*, pages 730–738. ACM/SIAM, 1997. URL: http://dl.acm.org/citation.cfm?id=314161.314429.

29 Kurt Mehlhorn. Pebbling moutain ranges and its application of dcfl-recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2_89.

30 Dan I Moldovan. *Parallel processing from applications to systems*. Elsevier, 2014.

31 V. Ramachandran. Restructuring formula trees. Unpublished manuscript, 1986.

32 Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

33 P.M. Spira. On time hardware complexity tradeoffs for boolean functions. *Proceedings of the Fourth Hawaii International Symposium on System Sciences*, pages 525–527, 1971.

34 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.

35 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

36 Heribert Vollmer. Introduction to circuit complexity: a uniform approach, 2013.

37 Egon Wanke. k-nlc graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2-3):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.

38 Barry Wilkinson and Michael Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, 1999.

Finding Pseudorandom Colorings of Pseudorandom Graphs *

Akash Kumar[1], Anand Louis[2], and Madhur Tulsiani[3]

1 **Purdue University, West Lafayette, USA**
akumar@purdue.edu
2 **Indian Institute of Science, Bangalore, India**
anandl@iisc.ac.in
3 **TTIC, Chicago, USA**
madhurt@ttic.edu

Abstract

We consider the problem of recovering a planted *pseudorandom* 3-coloring in expanding and low threshold-rank graphs. Alon and Kahale [SICOMP 1997] gave a spectral algorithm to recover the coloring for a random graph with a planted random 3-coloring. We show that their analysis can be adapted to work when coloring is pseudorandom i.e., all color classes are of equal size and the size of the intersection of the neighborhood of a random vertex with each color class has small variance. We also extend our results to partial colorings and low threshold-rank graphs to show the following:

- For graphs on n vertices with threshold-rank r, for which there exists a 3-coloring that is ε-pseudorandom and properly colors the induced subgraph on $(1-\gamma)\cdot n$ vertices, we show how to recover the coloring for $(1-O(\gamma+\varepsilon))\cdot n$ vertices in time $(r\cdot n)^{O(r)}$.
- For expanding graphs on n vertices, which admit a pseudorandom 3-coloring properly coloring all the vertices, we show how to recover such a coloring in polynomial time.

Our results are obtained by combining the method of Alon and Kahale, with eigenspace enumeration methods used for solving constraint satisfaction problems on low threshold-rank graphs.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph Coloring, Expanders, Spectral algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.37

1 Introduction

Given an undirected graph $G=(V,E)$, a k-coloring of G is a map $\chi: V\to[k]$ such that for all edges $\{u,v\}\in E$, we have $\chi(u)\neq\chi(v)$. Finding the minimum number of colors with which a graph can be colored, or even finding a coloring which uses few colors for a graph G which is promised to be 3-colorable has been a major open problem in the field of algorithm design. Starting from an early work of Wigderson [17] who showed how to color 3-colorable graphs with $O(\sqrt{n})$ colors, there has been a series of works using novel combinatorial ideas, as well as ideas based on semidefinite programming (SDP), which give algorithms for coloring 3-colorable graphs with fewer colors [6, 13, 7, 4, 9, 5, 14]. The most recent algorithm of

* The first author was supported by NSF CCF-1319080. Part of the work was done when he was a Project Associate at IISc during Summer 2017

[14] achieves a coloring with $o(n^{1/5})$ colors. On the hardness side, Dinur, Mossel and Regev [12] showed the hardness of coloring 3-colorable graphs with any constant number of colors, assuming a variant of the Unique Games Conjecture. It is also known [11, 15] to be NP-hard to find an independent set of size $n/9$ (which is a weaker goal than 9-coloring) in a graph G on n vertices, when G is promised to have a $(1-\varepsilon)$-*partial* 3-coloring i.e., a coloring which properly colors the induced subgraph on at least $(1-\varepsilon)\cdot n$ vertices.

There has also been a significant amount of research trying to recover a planted 3-coloring in special families of graphs [8, 1, 10]. Notably, an algorithm by Alon and Kahale [1], shows how to 3-color a random 3-colorable graph G (with high probability over the choice of G). The graphs in their model, denoted $\mathcal{G}_{3,p,n}$, are generated by dividing the vertices in three color classes of size $n/3$ each, and connecting each pair of vertices in distinct color classes independently with probability p. As their main result, [1] are able to recover the color classes in polynomial time with a small probability of failure even when p is as small as $3d/n$ for a constant d. Notice that in expectation a vertex in this graph will have degree $2d$ and will have d neighbors in each color class different from its own.

A generalization of the above result is to consider models with limited amount of randomness (semi-random) or arbitrary graphs with random-like properties (pseudorandom graphs). While both models seek to capture the minimal assumptions needed for the methods developed for random graphs, the study of pseudorandomness properties is also motivated by developing decompositions of worst-case objects into structured and pseudorandom parts [16]. The works by Blum and Spencer [8] already considered semi-random models for $p = n^{-(1-\delta)}$. Motivated by the notion of pseudorandomness and decompositions and pseudorandomness used in the sub-exponential algorithms for Unique Games [2, 3], Arora and Ge [5] showed how to find a large independent set in 3-colorable graphs with small threshold rank. A recent work of David and Feige [10] shows that even when the graph is pseudorandom (an expander) and the 3-coloring is arbitrary, one can recover the coloring on most vertices of the graph. They also show that when the coloring is random, it can be recovered for *all* vertices in the graph.

Our Results

In this work, we focus on showing that the first 2 steps in [1]'s proof can be adapted to (almost) 3-color a special family of pseudorandom graphs with a pseudorandom coloring (see Subsection 1.1 for a comparison with [10]). To state our results, we first define the relevant notions of pseudorandomness. Note that in the definitions below, we take the average degree of the graph to be $2d$, since we think of the degree in each color class as being d.

▶ **Definition 1** (Pseudorandom colorings)**.** Let $G = (V, E)$ be any graph with $|V| = n$ and $|E| = d \cdot n$. Let $\chi\colon V \to \{1, 2, 3\}$ be a (proper or improper) coloring of vertices with 3 colors. χ is considered $(2d, \varepsilon)$-pseudorandom if

1. χ is balanced i.e., all color classes ($1, 2, 3$ above) have the same size, $\frac{n}{3}$.
2. The coloring of G in the above step is a *low variance* coloring i.e., for all $i, j \in \{1, 2, 3\}$, we have $\mathrm{Var}_{v\in \mathrm{COL}_i} d_{ij}(v) \le \varepsilon \cdot d^2$.

The first family of pseudorandom graphs is low threshold-rank graphs, which (in the context of coloring) are defined as having a small number of negative eigenvalues with a large magnitude. This notion of pseudorandomness (with a different notion of the threshold) was also considered in the work of Arora and Ge [5].

▶ **Definition 2** (Threshold Rank)**.** A graph $G = (V, E)$ with $|E| = d \cdot n$. The threshold rank of G is defined to be the number of eigenvalues of the adjacency matrix that are smaller than $-9d/10$.

We will also require the following notion of a partial coloring.

▶ **Definition 3** (Partial Coloring)**.** For a graph $G = (V, E)$, a function $\chi : V \to [3] \cup \{\perp\}$ is said to be a $(1-\gamma)$-partial 3-coloring if

- $\left|\chi^{-1}(\perp)\right| \leq \gamma \cdot n$.
- χ is a proper 3-coloring for the induced subgraph on $\chi^{-1}([3])$.

Note that there exists a $(1-\gamma)$-partial 3-coloring χ of G if and only if there exits a *total* (but not necesarily proper) coloring $\chi' : V \to [3]$ such that χ' can be made a $(1-\gamma)$-partial 3-coloring by replacing the colors of γ fraction of vertices by $\perp$. Since our pseudorandomness properties are defined only for total colorings, we will abuse notation to say that there is a pseudorandom $(1-\gamma)$-partial 3-coloring if there exists a pseudorandom χ' which agrees with a $(1-\gamma)$-partial 3-coloring χ on $\chi^{-1}([3])$. We will refer to such a coloring as $(2d, \varepsilon)$-pseudorandom $(1-\gamma)$-partial 3-coloring. We now state the first theorem we prove.

▶ **Theorem 4.** *There exists an algorithm which, given $G = (V, E)$ such that*

- *$|E| = d \cdot n$ and $\mathsf{th}(G) = r$, and*
- *there exists χ, which is $(2d, \varepsilon)$-pseudorandom and forms a $(1-\gamma)$-partial 3-coloring of G,*

runs in time $\left(\frac{\sqrt{r \cdot n}}{\varepsilon}\right)^r \cdot poly(n)$ and w.h.p. returns a $(1 - O(\varepsilon + \gamma))$-partial 3-coloring of G.

We also consider the futher specialized pseudorandom family, where the graphs will also be required to be expanding. Note that the graphs below capture the random family considered by Alon and Kahale [1]. For graphs in this family, we will recover all the color classes exactly.

▶ **Definition 5** (Expanding 3-colorable graphs)**.** A 3-colorable graph is said to be expanding if for some small positive constant $\delta < 1$, $|\lambda_i| \leq \delta d$ holds $\forall\ 2 \leq i \leq n-2$ i.e., if all eigenvalues other than the leading eigenvalue and the last two are small in magnitude.

For graphs coming from this family, we have the following theorem.

▶ **Theorem 6.** *There exists a polynomial-time algorithm which, given $G = (V, E)$ such that*

- *G is an expanding 3-colorable graph with $|E| = d \cdot n$, and*
- *there exists $\chi : V \to [3]$, which is $(2d, \varepsilon)$-pseudorandom and a proper 3-coloring of G,*

recovers χ.

1.1 Related work

There is a huge amount of literature devoted to find a complete or partial (legal) 3-coloring of an input graph under some assumptions on the graph and/or some assumptions on some 3-coloring of the given graph. Here we briefly examine works related to the current work.

1.1.1 The algorithm of Alon and Kahale

[1] described how to 3-color a random graph $G \sim G_{3,d/n,n}$. Thus, they have a random graph with a planted 3-coloring which they 3-color using a three phase algorithm. Their main result shows that graphs coming from $G_{3,d/n,n}$ have nice spectral properties which, in the first phase, can be exploited by a spectral clustering approach to find a good candidate coloring, χ_1, which misclassfies very few vertices. This candidate coloring is later refined to obtain a coloring χ_2 in the next phase. The second phase is a local search which locally improves the coloring on the set $H \subseteq V$ of vertices which have close to d neighbors in each color class according to the current coloring. This is then followed up by a cleanup phase which

recolors vertices in $V \setminus H$ (all the components on the subgraph induced on these vertices have logarithmic sizes and can be brute forced upon).

We show how to construct χ_1 in low threshold-rank graphs which admit a pseudorandom partial coloring. When the graphs are also expanding admit a full coloring, we also show how to complete the second phase.

1.1.2 The results of David and Feige

[10] considered extensions of the problem of coloring random 3-colorable graphs in several directions. In particular, they tried to relax the randomness assumption on the graph and the planted coloring inherent in [1]. To this end, they considered 4 models. They take as input an approximately d-regular (spectrally) expanding graph which can be either *adversarial* or *random* and a balanced planted coloring which again can be *adversarial or random*.

Our results are somewhat incomparable with theirs. Perhaps the most directly related setting from their work is the one where the graph is an arbitrary expander with an arbitrary (balanced) 3-coloring. In this case, they recover a $(1-\gamma)$ partial coloring. In this work, we start with additional pseudorandomness assumptions on the coloring (beyond balance), and can recover a partial coloring without assuming expansion in the input graph G, but making a different assumption about the negative eigenvalues. When G is also expanding in addition to these properties, we can recover the coloring completely.

Notation

Our notations are standard. We will write vectors and matrices in boldface (like $\mathbf{u}$ and $\mathbf{A}$ respectievely). For a graph G we will denote its adjacency matrix by $\mathbf{A} = \mathbf{A}(G)$. We denote a unit vector along the direction of $\mathbf{u}$ with $\hat{\mathbf{u}}$. And the transpose of a vector $\mathbf{u}$ is denoted $\mathbf{u}^T$. Also, $\mathbf{1}$ will denote the vector which is 1 in every coordinate. The stationaty distribution of random walks will be denoted $\boldsymbol{\mu}$. We denote the degree of a vertex $i \in V$ by $deg(i)$. The set of edges with one end point in a set S and the other in T is denoted $E(S,T)$.

2 Partially 3-coloring partially 3-colorable graphs

As a first step, we use lemma 7, to get a full coloring which only miscolors an $O(\varepsilon+\gamma)$ fraction of the vertices. In the next step, we will uncolor the incorrectly colored vertices. The first step is summarized by the following lemma.

▶ **Lemma 7.** *There exists an algorithm which given a graph $G = (V, E)$ such that*

- $|E| = d \cdot n$ *and* $\mathsf{th}(G) = r$
- *there exists a coloring χ which is a $(2d, \varepsilon)$-pseudorandom coloring, and forms a $(1-\gamma)$-partial-3-coloring of G*

runs in time $O\left(\frac{\sqrt{r \cdot n}}{\varepsilon}\right)^r$ and returns a coloring which has $(1 - O(\varepsilon+\gamma))$ fraction of the vertices colored correctly, i.e., the graph induced on them has no monochromatic edges.

Following [1], we consider two special vectors which we call $\mathbf{x}$ and $\mathbf{y}$ as defined below.

$$\mathbf{x}(v) = \begin{cases} 2 & \text{if } v \in \mathrm{COL}_1, \\ -1 & \text{if } v \in \mathrm{COL}_2, \\ -1 & \text{if } v \in \mathrm{COL}_3, \end{cases} \qquad \mathbf{y}(v) = \begin{cases} 0 & \text{if } v \in \mathrm{COL}_1, \\ 1 & \text{if } v \in \mathrm{COL}_2, \\ -1 & \text{if } v \in \mathrm{COL}_3, \end{cases} \tag{1}$$

The point of these vectors is that they are both constant on all color classes and so are their linear combinations. Simlar to [1], we will try to find a vector which is close enough to some linear combination of $\mathbf{x}$ and $\mathbf{y}$ and use it to obtain a coloring of the kind Lemma 7 seeks. Now let us detail our algorithm.

Algorithm 1 Find Coloring

Require: Graph G with $\mathsf{th}(G) = r$ and the eigenvectors $\{\mathbf{v}_{n-r+1}, \mathbf{v}_{n-r+2}, \cdots \mathbf{v}_n\}$

1: Let $\mathbf{t} \leftarrow$ Subspace enumeration$(\sqrt{\frac{\varepsilon}{r}})$

2: Let $\mathrm{COL}_1 = \{\, i \in V : \mathbf{t}_i > 1/2 \,\}$, $\mathrm{COL}_2 = \{\, i \in V : \mathbf{t}_i < -1/2 \,\}$ and $\mathrm{COL}_3 = V \setminus (\mathrm{COL}_1 \cup \mathrm{COL}_2)$

This algorithm relies on a procedure called Subspace Enumeration which is described below.

Algorithm 2 Subspace enumeration(τ)

Require: Graph G with $\mathsf{th}(G) = r$ and the eigenvectors $\{\mathbf{v}_{n-r+1}, \mathbf{v}_{n-r+2}, \cdots \mathbf{v}_n\}$

1: Let $B_r \leftarrow [-100\sqrt{n}, 100\sqrt{n}]^r$

2: ▷ B_r denotes a bounding box in the space of r eigenvectors above

3: Partition B_r into grid cells. Each cell has length τ in all dimensions.

4: ▷ The number of cells produced is $O\left(\frac{200\sqrt{n}}{\tau}\right)^r$

5: Let P_r denote the set of all corners of any grid cell.

6: ▷ Thus, $P_r = \{\, \mathbf{p} \in B_r : \tau$ divides all r coordinates in $\mathbf{p} \,\}$

7: Find in P_r a point $\mathbf{t}$ which has

- $med(\mathbf{t}) = 0$ and $\|\mathbf{t}'\|_2 = \Theta(\sqrt{n})$
- distance at most $\leq O\left(\sqrt{\varepsilon n + \gamma n + \tau^2 r}\right)$ from $span(\mathbf{x}, \mathbf{y})$.

8: ▷ We later show, in Claim 10 that such a vector exists.

9: **return t**

To put this plan in motion, we make the following claims which are generalization of the corresponding claims in [1].

▶ **Claim 8.** $\|\mathbf{Ax} + d(1-\gamma)\mathbf{x}\|_2^2, \|\mathbf{Ay} + d(1-\gamma)\mathbf{y}\|_2^2 \leq O(\varepsilon n d^2)$.

▶ **Claim 9.** *There exist small shift vectors* $\mathbf{s}_n$ *and* $\mathbf{s}_{n-1}$ *with* $\|\mathbf{s}_n\|_2^2, \|\mathbf{s}_{n-1}\|_2^2 \leq O(\varepsilon n)$ *such that both* $\mathbf{x} - \mathbf{s}_n$ *and* $\mathbf{y} - \mathbf{s}_{n-1}$ *are the linear combinations of last* r *eigenvectors of* $\mathbf{A}$.

▶ **Claim 10.** *Algorithm 2 finds a vector* $\mathbf{t}$ *in the span of* $\{\mathbf{v}_{n-r+1}, \mathbf{v}_{n-r+2}, \ldots \mathbf{v}_n\}$ *in time* $O\left(\frac{\sqrt{n}}{\tau}\right)^r$ *such that*

- $med(\mathbf{t}) = 0$ *and* $\|\mathbf{t}'\|_2 = \Theta(\sqrt{n})$
- $\|\mathbf{t} - \mathbf{f}\|_2 \leq O(\sqrt{\varepsilon n + \gamma n})$ *where* $\mathbf{f}$ *is some vector that lies in* $span(\mathbf{x}, \mathbf{y})$.

Now, using these claims we will sketch how to establish Lemma 7. Taking cue from [1], we find a vector in the span of the last r eigenvectors, $span(\mathbf{v}_{n-r+1}, \mathbf{v}_{n-r+2}, \ldots, \mathbf{v}_n)$, a vector $\mathbf{t}$ which is close to a vector $\mathbf{f} \in span(\mathbf{x}, \mathbf{y})$ has length $\Theta(\sqrt{n})$ and median zero. In particular, the intuition is to have $\mathbf{t}$ (which the algorithm finds) be close to a vector $\mathbf{f}$ which has large positive entries indicating the first color class, large negative entries for the second color class and 0 entries for the last color class. In $\mathbf{t}$ hopefully the large positive entries and negative entries of $\mathbf{f}$ will remain away from zero and maintain their sign and the zero entries in $\mathbf{f}$ will hopefully remain close to 0. The remaining details of the proof are given in Appendix A. The proof follows [1].

With the proof of Lemma 7, let us see how to prove Theorem 4.

Proof. (Of Theorem 4) Let $E_{bad} \subseteq E$ be the set of edges which have both endpoints with the same color in the coloring derived using the first step described in the proof of Lemma 7. Let $U_{bad} \subseteq V$ be the set of endpoints of these edges and consider the graph $G[U_{bad}]$ induced on these vertices. We note that the set U of vertices misclassified in Lemma 7 also forms a vertex cover in $G[U_{bad}]$. By the standard 2-approximation algorithm for vertex cover we can find a vertex cover $C \subseteq U_{bad}$ where $|C| \leq 2|U|$. And on "uncoloring" the vertices in the set C we obtain a partial coloring which omits only a $O(|U|)$ of the vertices – which is a $(1 - O(\varepsilon + \gamma))$-partial 3-coloring. ◀

Now, let us prove Claim 8 and Claim 9.

Proof. (Of Claim 8) Let us prove this for the vector $\mathbf{x}$. The proof with vector $\mathbf{y}$ is similar. Let $\mathbf{u} = \mathbf{A}\mathbf{x} + d\mathbf{x}$. Let $\mathbf{a}_i^T$ denote the i^{th} row in $\mathbf{A}$. Thus, $\mathbf{u}_i = \mathbf{a}_i^T\mathbf{x} + d\mathbf{x}_i$. Let us say that in the vector $\mathbf{x}$, $\mathbf{x}_i = 2$ for $i \in \mathrm{COL}_1$, and it is -1 otherwise. Note that for $i \in \mathrm{COL}_1$, we get $\mathbf{u}_i^2 = (\mathbf{a}_i^T\mathbf{x} + d\mathbf{x})^2$

$$= (-d_{12}(i) - d_{13}(i) + 2d_{11}(i) + 2d(1-\gamma))^2 \leq O(M_1(i))$$

where $M_1(i) \stackrel{\text{def}}{=} (d_{12} - d)^2 + (d_{13} - d)^2 + (d_{11} - \gamma d)^2$. In a similar fashion, we see that for $i \in \mathrm{COL}_2$ $\mathbf{u}_i^2 \leq O(M_2(i))$ where $M_2(i) = (d_{21}(i) - d)^2 + (d_{23}(i) - d)^2 + (d_{22} - \gamma d)^2$. And an analogous upper bound holds for $\mathbf{u}_i^2$ when $i \in \mathrm{COL}_3$. Thus,

$$\|\mathbf{u}\|_2^2 = \sum \mathbf{u}_i^2 \leq \sum_{i \in \mathrm{COL}_1} M_1(i) + \sum_{i \in \mathrm{COL}_2} M_2(i) + \sum_{i \in \mathrm{COL}_3} M_3(i) \leq O\big(\varepsilon n d^2\big)$$

as the $\mathrm{Var}_{v \in \mathrm{COL}_i} d_{ii}(v) \leq \varepsilon d^2$ as well. The proof for upperbound on $\|\mathbf{A}\mathbf{y} + d\mathbf{y}\|_2^2$ is similar. ◀

And next, we prove Claim 9 as well. Here is a quick adaptation of [1]'s proof.

Proof. (Of Claim 9) Write $\mathbf{x} = \sum c_i \mathbf{v}_i$ in the eigenbasis. Again let $\mathbf{u} = \mathbf{A}\mathbf{x} + d\mathbf{x}$.

$$\|\mathbf{u}\|_2^2 = \sum_{i=1}^{n} c_i^2(\lambda_i + d(1-\gamma))^2 \geq \sum_{i=1}^{n-r} c_i^2(\lambda_i + d(1-\gamma))^2 \geq \Omega(d^2) \sum_{i=1}^{n-r} c_i^2$$

The last step above follows because the first $n - r$ eigenvalues are all at least $-9d/10$. And together with the fact that $\|\mathbf{u}\|_2^2 \leq O\big(\varepsilon n d^2\big)$, we get $\sum_{i=1}^{i=n-r} c_i^2 \leq O(\varepsilon n)$. And this is the shift vector $\mathbf{s}_n$ we seek with the desired bound on its length. A similar argument can be made to find $\mathbf{s}_{n-1}$ and its length using the vector $\mathbf{y}$. ◀

Proof of Claim 10. Let I denote the set of indices that are left uncolored in the hidden planted pseudorandom coloring. Let us define a vector $\mathbf{z}$ as $\mathbf{z}(i) = -1$ for $i \in \mathrm{COL}_1$, $\mathbf{z}(i) = 0$ for $i \in I \cup \mathrm{COL}_2$ and $\mathbf{z}(i) = +1$ for $i \in \mathrm{COL}_3$. Note that this there exists a vector (say $\mathbf{f}$) in $span(\mathbf{x}, \mathbf{y})$ such that $\mathbf{z}$ and $\mathbf{f}$ are pretty close – the distance is at most $\sqrt{\gamma n}$. Further, there also exists a vector $\mathbf{t} \in span(\mathbf{v}_{n-r+1}, \mathbf{v}_{n-r+2}, \cdots, \mathbf{v}_n)$ which is pretty close to $\mathbf{z}$. This distance is easily bounded by arguments similar to Claim 8 and Claim 9. In particular, it follows that the distance of $\mathbf{z}$ and $\mathbf{t}$ considered above is at most $O(\sqrt{\varepsilon n + \gamma n})$. And therefore, the distance $\|\mathbf{t} - \mathbf{f}\|_2 \leq O(\sqrt{\varepsilon n + \gamma n})$. And finally, we also note that $med(\mathbf{t}) = 0$ and $\|\mathbf{t}\|_2 = \Theta(\sqrt{n})$.

This only proves that such a vector exists. We need to algorithmically *find* one. To this end, we use the *subspace enumeration* procedure described earlier. Given any vector $\mathbf{p} \in B_r$ (which, recall, was defined in Algorithm 2), this procedure finds a vector $\mathbf{t}$ which is within

a distance $\tau\sqrt{r}$ of $\mathbf{p}$. In particular, this also holds for $\mathbf{t}'$. Moreover, $\mathbf{t}'$ being close to $\mathbf{z}$ has several 0 entries one of which is the median. And the vector $\mathbf{t}$ being $\tau\sqrt{r}$ close to $\mathbf{t}'$ also has $med(\mathbf{t}) = 0$, length $\Theta(\sqrt{n})$ and has $\|\mathbf{t} - \mathbf{f}\|_2 \leq O\left(\sqrt{\varepsilon n + \gamma n + \tau^2 r}\right)$ as can be easily seen by triangle inequality. ◀

3 Coloring expanding-3-colorable

In the case of $(2d, \varepsilon)$-expanding-3-colorable, $\gamma = 0$. So, the coloring obtained by the first step of the algorithm returns a $(1 - O(\varepsilon))$ partial coloring. We briefly review what [1] do in the second step for 3-coloring a random 3-colorable graph $G \in G_{3,d/n,3n}$. Their algorithm receives as input a 3-colored graph G and a set U of *bad* vertices (with $|U| = O(n/d)$) that have been incorrectly colored. This bad 3 coloring of G is improved via an iterative process. Each step in the iteration reduces the number of bad vertices by a constant factor. The algorithm is given below, and we use the same algorithm. However, our analysis is different.

Algorithm 3 Improve Coloring

Require: A $(1 - O(\varepsilon))$partial 3-coloring of G vertices.

1: Let the current color classes be denoted V_1^0, V_2^0 and V_3^0
2: Add the uncolored vertices arbitrarily to the partitions in any order.
3: **for** $i = 1 \colon \log n$ **do**
4: For $j \in \{1, 2, 3\}$, put $v \in V_j^i$ only if $|N(v) \cap V_j^{i-1}| \leq |N(v) \cap V_l^{i-1}|$ for all $l \neq j$
5: ▷ i.e.,, put v in the least popular color class among its neighbors from previous iteration
6: ▷ V_1^i, V_2^i, V_3^i denotes the color classes in the i^{th} iteration
7: **end for**

The above algorithm derives from the following intuition. Given a 3-colored graph $G \in G_{3,d/n,3n}$ with a small set U of bad vertices, a "local search procedure" can recolor those vertices in V which do not have way too many neighbors in U. Thus, another way to express the intuition is to say that a bad vertex at the beginning of iteration i remains bad after iteration i finishes only if it is surrounded by many bad vertices. To make this formal, let us consider the set

$$W = \{\, v \in V \,:\, deg_U(v) \geq d/4 \,\}.$$

W will be referred to as being U*-rich*. The main step in the argument is to show that the size of U-rich set is at most $\frac{|U|}{2}$ and that at the end of every iteration the set which remains incorrectly colored is only a subset of the U-rich set. We will now use this argument in the case of expanding graphs. This is done in the following lemma.

▶ **Lemma 11.** *There exists a polynomial time algorithm which returns, given* $(2d, \varepsilon)$-expanding-3-colorable *graph* $G = (V, E)$ *(recall that this means for some absolute positive constant* $\delta < 1$, $|\lambda_i| \leq \delta d$ *for* $2 \leq i \leq n - 2$*) and a* $(1 - O(\varepsilon))$ *partial* 3*-coloring of* V*, a proper* 3 *coloring of* V*.*

We will prove Lemma 11 by using the expander mixing lemma. The key here would be to again show that $|U|$-rich sets have small size if $|U|$ is small. This is done in the following claim.

▶ **Claim 12.** *Let* $U \subseteq V$ *be a set with size at most* $O(\varepsilon n)$*. Let*

$$W = \{\, v \in V \,:\, deg_U(v) \geq 2d/5 \,\}.$$

Then $|W| \leq 0.99|U|$.

Proof of Claim 12. We begin by finding upper bounds and lower bounds for $|E(U,W)|$. We see that

$$|E(U,W)| \geq \frac{1}{2} \cdot \frac{2d|W|}{5} = \frac{d|W|}{5}.$$

The inequality here comes because there are $\frac{2d}{5}$ edges incident on every vertex in $|W|$ with at least one endpoint in U. The factor of $1/2$ compensates for the fact that both the endpoints of the edge may belong to U.

Now, we need to upper bound $|E(U,W)|$. To do this, we use expander mixing lemma. Let the indicator vector for U be denoted $\mathbb{1}_{\mathbf{U}}$ and the indicator vector for W be denoted $\mathbb{1}_{\mathbf{W}}$. Let us write $\mathbb{1}_{\mathbf{U}} = \sum \alpha_i \mathbf{v}_i$ and $\mathbb{1}_{\mathbf{W}} = \sum \beta_i \mathbf{v}_i$. We know

$$|E(U,W)| = {\mathbb{1}_{\mathbf{U}}}^T \mathbf{A} \mathbb{1}_{\mathbf{W}} = |\left(\sum \alpha_i \mathbf{v}_i^T\right) \mathbf{A} \left(\sum \beta_j \mathbf{v}_j\right)| = |\sum (\alpha_i \beta_i \lambda_i)|$$

And it is immediately seen that the following holds

$$|E(U,W)| \leq |\alpha_1\beta_1\lambda_1| + |\alpha_n\beta_n| \cdot |\lambda_n| \ + |\alpha_{n-1}\beta_{n-1}| \cdot |\lambda_{n-1}| + \sum_{i=2}^{n-2} |\alpha_i\beta_i\lambda_i| \tag{2}$$

Now, we upper bound the RHS of Equation 2. We show how to bound each of the summands seprately.

1. Let us begin by bounding the first summand. For intuition sake consider the $\varepsilon = 0$ case. Then in fact we have a $2d$ regular graph and $\mathbf{v}_1 = \mathbf{u}_n$ where $\mathbf{u}_n = \frac{1}{\sqrt{n}} \cdot \mathbf{1}$ denotes the normalized uniform distribution vector.

$$\begin{aligned} |\alpha_1\beta_1\lambda_1| \leq & |{\mathbb{1}_{\mathbf{U}}}^T \mathbf{v}_1| \cdot |{\mathbb{1}_{\mathbf{W}}}^T \mathbf{v}_1| \cdot 2d \\ = & \frac{|U|}{\sqrt{n}} \cdot \frac{|W|}{\sqrt{n}} \cdot 2d = \frac{|U| \cdot |W| \cdot 2d}{n} \end{aligned} \tag{3}$$

   In case $\varepsilon > 0$, we will show that something close holds. In particular, we will show that the vector $\mathbf{v}_1$ has a large dot product with $\mathbf{u}_n$. Denote by $\boldsymbol{\mu}$ the stationary distribution vector for random walks on G (and thus, $\boldsymbol{\mu}_i = \frac{deg(i)}{\sum deg(i)}$). Now, note

$$\begin{aligned} \left(\frac{\mathbf{v}_1^T \mathbf{1}}{\sqrt{n}}\right)^2 = \left(\frac{1}{\sqrt{n}} \cdot \left(\frac{\boldsymbol{\mu}}{\|\boldsymbol{\mu}\|_2}\right)\right)^2 = \frac{1}{n} \cdot \frac{(\sum \boldsymbol{\mu}_i)^2}{\|\boldsymbol{\mu}\|_2^2} = \frac{1}{n} \cdot \frac{1}{\|\boldsymbol{\mu}\|_2^2} = \frac{(\sum deg(i))^2}{n \sum deg(i)^2} \\ = \frac{(\mathbb{E}[deg(i)])^2}{\mathbb{E}[deg(i)^2]} = \frac{1}{1 + \frac{\mathrm{Var}_{i \sim V}\, deg(i)}{\mathbb{E}[deg(i)]^2}} \geq \frac{1}{1 + 4\varepsilon d^2/d^2} = \frac{1}{1+4\varepsilon} \geq 1 - 8\varepsilon \end{aligned}$$

   Here the first inequality follows because of Claim 13 which is proved later in this section.

   ▶ **Claim 13.** $\mathrm{Var}_{i \sim V}\, deg(i) \leq 4\varepsilon d^2$.

   This means that the vector $\mathbf{v}_1$ is very close to the vector $\mathbf{1}/\sqrt{n}$. Now, we know that vector ${\mathbb{1}_{\mathbf{U}}}^T \frac{\mathbf{1}}{\sqrt{n}}$ is small. And we know that $\frac{\mathbf{v}_1^T \mathbf{1}}{\sqrt{n}}$ is large. It follows from this that ${\mathbb{1}_{\mathbf{U}}}^T \mathbf{v}_1$ will also be fairly small. In particular, we get

$$|\alpha_1\beta_1\lambda_1| \leq 5\frac{|U|}{\sqrt{n}}(1+8\varepsilon) \cdot \frac{|W|}{\sqrt{n}}(1+8\varepsilon) \cdot 2d \leq \frac{|U| \cdot |W| \cdot 2d(1+20\varepsilon)}{n} \tag{4}$$

   Now, we bound the second summand.

   As we will see in Claim 15, we have $\sqrt{n}\mathbf{v}_n$ can be expressed as a linear combination of vectors $\mathbf{x} - \mathbf{s}_n$ and $\mathbf{y} - \mathbf{s}_{n-1}$ with coefficients $O(1)$ in absolute value. So, let us write $\sqrt{n}\mathbf{v}_n = a_1(\mathbf{x} - \mathbf{s}_n) + b_1(\mathbf{y} - \mathbf{s}_{n-1})$ with a_1 and b_1 being constants. A similar expression can be obtained for $\sqrt{n}\mathbf{v}_{n-1}$. Now, we claim

▶ **Claim 14.** $|\alpha_n\beta_n\lambda_n| \leq O\left(d \cdot \left(\frac{|U||W|}{n} + \frac{|U|\sqrt{|W|}\sqrt{\varepsilon}}{\sqrt{n}} + \frac{|W|\sqrt{|U|}\sqrt{\varepsilon}}{\sqrt{n}} + \varepsilon\sqrt{|U||W|}\right)\right)$

Proof of Claim 14.

$$\begin{aligned}|\alpha_n\beta_n\lambda_n| &\leq \frac{1}{\sqrt{n}}|\mathbb{1}_{\mathbf{U}}{}^T\sqrt{n}\mathbf{v}_n| \cdot \frac{1}{\sqrt{n}}|\mathbb{1}_{\mathbf{W}}{}^T\sqrt{n}\mathbf{v}_n| \cdot 2d \\ &\leq \frac{|\mathbb{1}_{\mathbf{U}}{}^T a_1(\mathbf{x}-\mathbf{s}_n) + \mathbb{1}_{\mathbf{U}}{}^T b_1(\mathbf{y}-\mathbf{s}_{n-1})|}{\sqrt{n}} \\ &\quad \times \frac{|\mathbb{1}_{\mathbf{W}}{}^T a_2(\mathbf{x}-\mathbf{s}_n) + \mathbb{1}_{\mathbf{W}}{}^T b_2(\mathbf{y}-\mathbf{s}_{n-1})|}{\sqrt{n}} \cdot 2d \end{aligned} \tag{5}$$

We now need to understand how to bound terms like $\frac{1}{\sqrt{n}}|\mathbb{1}_{\mathbf{U}}{}^T a_1(\mathbf{x}-\mathbf{s}_n)|$. To this end, note that we have the following.

a. $|\mathbb{1}_{\mathbf{U}}{}^T a_1(\mathbf{x}-\mathbf{s}_n)| \leq |\mathbb{1}_{\mathbf{U}}{}^T a_1\mathbf{x}| + |a_1|\|\mathbf{s}_n\|_2\|\mathbb{1}_{\mathbf{U}}\|_2 \leq |\mathbb{1}_{\mathbf{U}}{}^T a_1\mathbf{x}| + |a_1|\sqrt{\varepsilon}\sqrt{n}\sqrt{|U|}$.

b. $|\mathbb{1}_{\mathbf{U}}{}^T\mathbf{x}| = |2U_1 - U_2 - U_3| \leq 2|U|$
Here U_1 refers to number of vertices colored with COL_1 in U. Other terms have analogous meaning.

Putting all of this together and absorbing all constants in the $O(.)$, we get

$$|\alpha_n\beta_n\lambda_n| \leq O\left(d \cdot \left(\frac{|U|}{\sqrt{n}} + \sqrt{\varepsilon}\sqrt{|U|}\right) \cdot \left(\frac{|W|}{\sqrt{n}} + \sqrt{\varepsilon}\sqrt{|W|}\right)\right) \tag{6}$$

◀

2. The $3rd$ item has an analogous bound to the one above. That is, we have

$$|\alpha_{n-1}\beta_{n-1}\lambda_{n-1}| \leq O(|\alpha_n\beta_n\lambda_n|) \tag{7}$$

3. The last item can be simply bounded by using Cauchy Schwartz and the fact that all other eigenvalues are at most δd for some sufficiently small δ. This gives

$$\sum_{i=1}^{n-2} |\alpha_i\beta_i\lambda_i| \leq \delta d \cdot \sqrt{|U| \cdot |W|} \tag{8}$$

Putting all of these bounds Equation 4, Equation 6, Equation 7, Equation 8 together we get and upper bound on $|E(U,W)|$. Also, we already computed a lower bound on $|E(U,W)| \geq d|W|/5$. Chaining the upper bound and the lower bound thus obtained, using the fact that $|U| \leq O(\varepsilon n)$ we get

$$\begin{aligned}\frac{\sqrt{|W|}}{5} &\leq \frac{2|U|\sqrt{|W|}(1+20\varepsilon)}{n} + O\left(\frac{|U|\sqrt{|W|}}{n} + \frac{\sqrt{\varepsilon|U||W|}}{\sqrt{n}} + \frac{\sqrt{\varepsilon}|U|}{\sqrt{n}} + \varepsilon\sqrt{|U|}\right) \\ &\quad + \delta\sqrt{|U|} \\ \implies \frac{\sqrt{|W|}}{10} &\leq O\Big(\varepsilon\sqrt{|U|} + \delta\sqrt{|U|}\Big) \\ \implies |W| &\leq O\big(\delta^2 + \varepsilon^2\big)|U|.\end{aligned}$$

This finishes the proof of Claim 12. ◀

Proof of Lemma 11. Observe that by Claim 12, it follows that the set of badly colored vertices shrinks significantly in each step of the local search. Thus, after $O(\log |U|)$ many steps, we do not have any bad vertices and we get a proper 3-coloring. This finishes the description of the algorithm for adapting the second step of [1] algorithm which finishes the proof of Lemma 11 ◀

Proof of Claim 13. Note that here we are taking variance in the degree of a random vertex over the full graph whereas by definition in $(2d, \varepsilon)$-expanding-3-colorable graph case we only know $\mathrm{Var}_{v \in \mathrm{COL}_j} d_{ij}(v) \leq \varepsilon d^2$. So, a little more work is needed. Let us begin by noting that $\mathbb{E}\big[deg(i)^2\big] = \sum_{j \in \{1,2,3\}} \mathbb{E}\big[deg(i)^2 | i \in \mathrm{COL}_j\big]$ /3. Also, note $\mathbb{E}[degi | i \in \mathrm{COL}_1] = \mathbb{E}[deg(i) | i \in \mathrm{COL}_2] = \mathbb{E}[deg(i) | i \in \mathrm{COL}_3] = d$. Let $M_1 = \mathbb{E}_{i \in \mathrm{COL}_1}\left[(d_{12}(i) + d_{23})^2 \right]$ and $N_1 = \mathbb{E}_{i \in \mathrm{COL}_1}\left[(d_{12}(i)^2 + d_{13}(i)^2) \right]$ and similarly define M_2, N_2 and M_3, N_3. So, writing $\mathrm{Var}_{i \sim V}\, deg(i) = \mathbb{E}\big[deg(i)^2\big] - \mathbb{E}[deg(i)]^2$ and expanding by using the foregoing expressions, we get

$$\begin{aligned}
\mathrm{Var}_{i \sim V} &= \sum_{j \in \{1,2,3\}} \frac{\mathbb{E}\big[deg(i)^2 | i \in \mathrm{COL}_j\big] - 4d^2}{3} \\
&= \sum_{j \in \{1,2,3\}} \frac{M_j - 4d^2}{3} \\
&\leq \sum_{j \in \{1,2,3\}} \frac{2N_j - 4d^2}{3} \\
&= \frac{2(\mathrm{Var}\, d_{12} + \mathrm{Var}\, d_{13} + \ldots \mathrm{Var}\, d_{32} + \mathrm{Var}\, d_{31})}{3} \\
&\leq 4\varepsilon d^2
\end{aligned}$$

◀

Acknowledgements. We thank Elena Grigorescu for helpful discussions.

References

1. Noga Alon and Nabil Kahale. A spectral technique for coloring random 3-colorable graphs. *SIAM J. Comput.*, 26(6):1733–1748, 1997.
2. Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 563–572. IEEE, 2010.
3. Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM (JACM)*, 62(5):42, 2015.
4. Sanjeev Arora and Eden Chlamtac. New approximation guarantee for chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 215–224, 2006.
5. Sanjeev Arora and Rong Ge. New tools for graph coloring. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - APPROX 2011 Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 1–12, 2011.
6. Avrim Blum. New approximation algorithms for graph coloring. *J. ACM*, 41(3):470–516, 1994.
7. Avrim Blum and David R. Karger. An õ(n^{3/14})-coloring algorithm for 3-colorable graphs. *Inf. Process. Lett.*, 61(1):49–53, 1997.
8. Avrim Blum and Joel Spencer. Coloring random and semi-random k-colorable graphs. *J. Algorithms*, 19(2):204–234, 1995.

9 Eden Chlamtac. Approximation algorithms using hierarchies of semidefinite programming relaxations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 691–701, 2007.

10 Roee David and Uriel Feige. On the effect of randomness on planted 3-coloring models. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 77–90, 2016.

11 Irit Dinur, Subhash Khot, Will Perkins, and Muli Safra. Hardness of finding independent sets in almost 3-colorable graphs. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 212–221. IEEE, 2010.

12 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM Journal on Computing*, 39(3):843–873, 2009.

13 David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998.

14 Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *J. ACM*, 64(1):4:1–4:23, 2017.

15 Subhash Khot and Rishi Saket. Hardness of finding independent sets in almost q-colorable graphs. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 380–389. IEEE, 2012.

16 Terence Tao. *Structure and randomness: pages from year one of a mathematical blog.* American Mathematical Soc., 2008.

17 Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM (JACM)*, 30(4):729–735, 1983.

A Missing Proofs

Proof of Lemma 7. Obtain vectors $\mathbf{t}$ and $\mathbf{f}$ from Claim 10. These vectors have several useful properties which will be useful for us in what follows. Write $\mathbf{t} = \mathbf{f} + \mathbf{w}$. We have $\|\mathbf{w}\|_2 \leq O\Big(\sqrt{\varepsilon n + \gamma n + \tau^2 r}\Big)$.

Now, let $\alpha_i = \mathbf{f}\big|_{\mathrm{COL}_i}$ for $1 \leq i \leq 3$ – that is, α_i's are scalars and equal the constant value that $\mathbf{f}$ takes over the i^{th} color class. Assume $\alpha_1 \geq \alpha_2 \geq \alpha_3$. Using the fact that $\|\mathbf{w}\|^2 \leq O(\varepsilon n + \gamma n + \tau^2 r)$, it follows that only $O\big(\varepsilon n + \gamma n + \tau^2 r\big)$ coordinates in $\mathbf{w}$ can take on values large in magnitude (at least 0.01). Note that this means $|\alpha_2| \leq 1/4$. This is because, we have $\mathbf{t} - \mathbf{f} = \mathbf{w}$ and only a few entries in $\mathbf{w}$ can be big in magnitude. In more detail, suppose if $\alpha_2 > 1/4$ were to hold then at least $2n - O\big(\varepsilon n + \gamma n + \tau^2 r\big)$ entries in $\mathbf{t}$ will be big too and thus will have the same sign. This contradicts the 0 median assumption in $\mathbf{t}$. An analogous argument prevents $\alpha_2 < -1/4$.
Finally, recall that $\alpha_1 + \alpha_2 + \alpha_3 = 0$ (since $\mathbf{f} \in span(\mathbf{x}, \mathbf{y})$) and

$$\alpha_1^2 + \alpha_2^2 + \alpha_3^2 = \frac{\|\mathbf{f}\|^2}{n/3} \geq \frac{\|\mathbf{t}\|^2 - \|\mathbf{w}\|^2}{n/3} \geq 6 - O\big(\varepsilon + \gamma + \tau^2 r/n\big).$$

Together with the fact that $|\alpha_2| \leq 1/4$, this implies that $\alpha_1 \geq 3/4$ and $\alpha_3 \leq -3/4$. This means that $\mathbf{t}$ which is obtained by corrupting $\mathbf{f}$ on at most $O(\varepsilon n)$ entries coming from the noisy vector is a pretty good coloring. And the number of misclassified vertices in the coloring according to $\mathbf{t}$ is at most $O(\varepsilon n + \gamma n + \tau^2 r)$ – the ℓ_2^2 length of the noisy vector $\mathbf{w}$.

And now note that setting $\tau = \sqrt{\frac{\varepsilon}{r}}$ prodcues $\mathbf{w}$ with $\|w\|_2^2 = O(\varepsilon n + \gamma n)$. And the size of the discrete subspace that we search over is at most $O\Big(\frac{\sqrt{r \cdot n}}{\varepsilon}\Big)^r$ as claimed. And this finishes the proof. ◀

▶ **Claim 15.** *The vectors $\sqrt{n}\mathbf{v}_n$ and $\sqrt{n}\mathbf{v}_{n-1}$ can be expressed as a linear combination of vectors $\mathbf{x}-\mathbf{s}_n$ and $\mathbf{y}-\mathbf{s}_{n-1}$ with coefficients at most a constant in absolute value.*

Proof of Claim 15. Let us first quickly see that $\mathbf{v}_n$ lies in $span(\mathbf{x}-\mathbf{s}_n, \mathbf{y}-\mathbf{s}_{n-1})$. By definition of $(2d,\varepsilon)$-expanding-3-colorable graphs, we have that all eigenvalues other than λ_1, λ_{n-1} and λ_n are small in absolute value. So, there exist tiny shift vectors $\mathbf{s}_n, \mathbf{s}_{n-1}$ (according to Claim 9) such that both $\mathbf{x}-\mathbf{s}_n, \mathbf{y}-\mathbf{s}_{n-1}$ lie in $span(\mathbf{v}_n, \mathbf{v}_{n-1})$. Thus, the space spanned by the last 2 eigenvectors is a tiny perturbation of $span(\mathbf{x},\mathbf{y})$. So, any vector in $span(\mathbf{v}_n, \mathbf{v}_{n-1})$ also lies in $span(\mathbf{x}-\mathbf{s}_n, \mathbf{y}-\mathbf{s}_{n-1})$. And so, does $\sqrt{n}\mathbf{v}_n$.

Intuitively, the at most constant in absolute value part should follow because $\mathbf{x}-\mathbf{s}_n, \mathbf{y}-\mathbf{s}_{n-1}, \sqrt{n}\mathbf{v}_n, \sqrt{n}\mathbf{v}_{n-1}$ have length $\Theta(\sqrt{n})$ and $\mathbf{x}-\mathbf{s}_n, \mathbf{y}-\mathbf{s}_{n-1}$ are nearly orthogonal. This is because $\mathbf{x}$ and $\mathbf{y}$ are orthogonal and $\mathbf{s}_n$ and $\mathbf{s}_{n-1}$ are very tiny perturbations to these orthogonal vectors. In more detail,

$$\begin{aligned} &\sqrt{n}\mathbf{v}_n = \alpha(\mathbf{x}-\mathbf{s}_n) + \beta(\mathbf{y}-\mathbf{s}_{n-1}) \\ \implies &\alpha\mathbf{x}+\beta\mathbf{y} = \sqrt{n}\mathbf{v}_n + \alpha\mathbf{s}_n + \beta\mathbf{s}_{n-1} \\ \implies &\|\alpha\mathbf{x}+\beta\mathbf{y}\|_2 \leq \sqrt{n} + |\alpha|\cdot\|\mathbf{s}_n\|_2 + |\beta|\cdot\|\mathbf{s}_{n-1}\|_2 \end{aligned} \tag{9}$$

On taking squared ℓ_2 norms on both sides of (9), it follows that

$$\frac{6n\alpha^2}{3} + \frac{2n\beta^2}{3} \leq n + O(\alpha^2+\beta^2)\varepsilon n.$$

On simplifying it is clear that this means both $|\alpha|$ and $|\beta|$ are $O(1)$. ◀

Flow Games*

Orna Kupferman[†1], Gal Vardi[‡2], and Moshe Y. Vardi[§3]

1 **School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel**
2 **School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel**
3 **Department of Computer Science, Rice University, Houston, Texas, USA**

Abstract

In the traditional maximal-flow problem, the goal is to transfer maximum flow in a network by directing, in each vertex in the network, incoming flow into outgoing edges. While the problem has been extensively used in order to optimize the performance of networks in numerous application areas, it corresponds to a setting in which the authority has control on all vertices of the network. Today's computing environment involves parties that should be considered adversarial. We introduce and study *flow games*, which capture settings in which the authority can control only part of the vertices. In these games, the vertices are partitioned between two players: the authority and the environment. While the authority aims at maximizing the flow, the environment need not cooperate. We argue that flow games capture many modern settings, such as partially-controlled pipe or road systems or hybrid software-defined communication networks. We show that the problem of finding the maximal flow as well as an optimal strategy for the authority in an acyclic flow game is Σ_2^P-complete, and is already Σ_2^P-hard to approximate. We study variants of the game: a restriction to strategies that ensure no loss of flow, an extension to strategies that allow non-integral flows, which we prove to be stronger, and a dynamic setting in which a strategy for a vertex is chosen only once flow reaches the vertex. We discuss additional variants and their applications, and point to several interesting open problems.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Flow networks, Two-player Games, Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.38

1 Introduction

A *flow network* is a directed graph in which each edge has a capacity, bounding the amount of flow that can go through it. The amount of flow that enters a vertex equals the amount of flow that leaves it, unless the vertex is a *source*, which has only outgoing flow, or a *target*, which has only incoming flow. The fundamental *maximum-flow problem* gets as input a flow network with a source vertex and a target vertex and searches for a maximal flow from the

* A full version of the paper is available at `https://www.dropbox.com/s/4z3775pjy4q4h44/all.pdf?dl=0`.

† 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)

‡ 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)

§ Work supported in part by NSF grants CCF-1319459 and IIS-1527668, and by NSF Expeditions in Computing project "xCAPE: Expeditions in Computer Augmented Program Engineering"

source to the target [9, 19]. The problem was first formulated and solved in the 1950's [16, 17]. It has attracted much research on improved algorithms [11, 10, 20] and applications [2].

The maximum-flow problem can be applied in many settings in which something travels along a network. This covers numerous applications domains, including traffic in road or rail systems, fluids in pipes, currents in an electrical circuit, packets in a communication network, and many more [2]. Less obvious applications involve flow networks that are constructed in order to model settings with an abstract network, as in the case of elimination in partially completed tournaments [32] or scheduling with constraints [2]. In addition, several classical graph-theory problems can be reduced to the maximum-flow problem. This includes the problem of finding a maximum bipartite matching, minimum path cover, maximum edge-disjoint or vertex-disjoint path, and many more [9, 2]. Variants of the maximum-flow problem can accommodate further settings, like circulation problems, where there are no sink and target vertices, yet there is a lower bound on the flow that needs to be traversed along each edge [34], networks with multiple source and target vertices [12], networks with costs for unit flows, and more.

All studies of flow networks so far assume that all the vertices in the network can be controlled by a central authority. That is, the maximum-flow algorithm finds a flow that directs, in all vertices of the network, incoming flow into the outgoing edges. In many applications of flow networks, however, only some of the vertices of the network can be so controlled: In road systems, police can direct traffic in only some of the junctions; in pipe systems, a company may have control only on some of the valves; and in communication networks, the authority may control only in some of the routers. In particular, in the area of *software defined network* (SDN), there is growing interest in hybrid networks, where some vertices are software-defined by a logically-centralized controller and in some others the routers behave as they see fit (e.g., running traditional or proprietary routing protocols, making adversarial decisions, etc.). Moreover, new network elements (e.g., SDN switches) should be added to the network gradually, so that traffic in not suspended, which results in a hybrid network [1, 35].

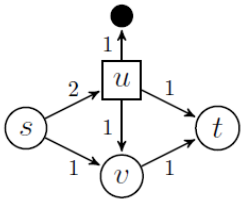
The above applications suggest that the maximal-flow problem should be revisited, taking into account the *game-theoretic* nature of the setting. Indeed, in more and more applications, the authority can direct the flow only in a subset of the vertices. In the others, it is the environment that directs the flow. The environment need not cooperate, giving rise to a game between the authority and the environment.

We introduce and study *flow games*.[1] In a flow game, the vertices in the network are partitioned between two players, Player 0 and Player 1. Player 0 corresponds to the network authority, whose goal is to maximize the flow, while Player 1 corresponds to the hostile environment. A strategy for a player advices him how to direct flow that enters vertices under his control. Formally, for each vertex u, let E_u denote the set of edges outgoing from u. Also, for each edge e, let $c(e) \in \mathbb{N}$ denote its capacity. Then, for each vertex u controlled by the player, a strategy for the player includes a policy $f_u : \mathbb{N} \to \mathbb{N}^{E_u}$ that maps every incoming flow $x \in \mathbb{N}$ to a function describing how x is partitioned among the edges outgoing from u. For each incoming flow $x \in \mathbb{N}$ and edge $e \in E_u$, we require that $f_u(x)(e) \leq c(e)$ and $\sum_{e \in E_u} f_u(x)(e) = \min\{x, \sum_{e \in E_u} c(e)\}$. Thus, $f_u(x)$ assigns to each edge outgoing from u a flow that is bounded by its capacity. Also, when the incoming flow is larger than the capacity of the outgoing edges (which bounds the outgoing flow), then flow *gets stuck* or *leaks* and the

[1] Not to confuse with games in which players *cooperate* in order to construct a sub-graph that maximizes the flow in the traditional setting, which are also termed flow games (c.f., [24]).

outgoing flow is lower than the incoming flow. In addition, a policy for the source s assigns to each edge outgoing from s a flow that is bounded by its capacity. The goal of Player 0 is to maximize the flow that enters the target t, no matter how Player 1 plays. Thus, it is a *Stackelberg game* where Player 0 is the leader [29]. Note that the definition of flow in a flow game is different from the traditional definition, which corresponds to the case all vertices belong to Player 0, and in which the "flow conservation" property is respected in all vertices. Indeed, in the game setting, Player 1 may cause flow to gets stuck or to leak by directing the flow to vertices whose outgoing capacity is smaller than the incoming flow.

▶ **Example 1.** Consider the flow game in the figure below. We represent vertices of Player 0 by circles and vertices of Player 1 by squares. Sinks, namely vertices other than the target that do not have outgoing edges, are represented by filled circles. In the classical max-flow problem, the maximal flow is 2, which is also the minimal cut in the network.



We can view the network as a road system, where the vertex u, which belongs to Player 1, is a junction in which the police does not direct incoming traffic. Note that the traffic is not lost in u, it is only that the police cannot direct it to t, and so it may go to the sink, where it gets lost, or to vertex v. In the context of road systems, flow loss means that the outgoing flow is less than the incoming flow and thus a traffic jam occurs. Likewise, the network may model a communication network in which the vertex u is a router whose software we do not control. Again, unless the outgoing channels are filled, the router does not dismiss packets that reach it, but it can direct the packets however it chooses. A strategy for Player 0 that directs 1 unit of flow from s to v and no flow from s to u ensures a flow of 1. Also, since the incoming flow to u is at most 2, and Player 1 can direct it to the sink and to vertex v, Player 0 does not have a strategy that ensures a flow of more than 1. Hence, the maximum flow that Player 0 can ensure is 1.

In essence, what we propose here is to lift the maximum-flow problem from its classical *one-player* setting to a *two-player* setting. Such a transition has been studied in computer science in many contexts. In graph theory, this transition corresponds to going from graph reachability to alternating graph reachability (*Path Systems*) [7]. In complexity theory, this is the transition from nondeterministic computation to alternating computation [6]. In logic, this is the transition from Boolean satisfiability [8] to quantified Boolean satisfiability [33]. In temporal reasoning, this is the transition from satisfiability [27] to temporal synthesis [30]. Generally, this can be viewed as a transition from *closed systems*, which are completely under our control, to *open systems*, in which we have to contend with adversarial environments. The absence of regulation by some central authority is indeed a driving theme of *algorithmic game theory*, cf. [28], inspired by the open nature of today's computing environments. Thus, this work can be viewed as a study of maximal flow in a network from the perspective of algorithmic game theory.[2] In addition, flow games extend less direct applications of the

[2] Different aspects of networks have already been extensively studied from the perspectives of algorithmic

maximum-flow problem to open settings. In particular, many of the constrained scheduling problems that are reduced to maximum-flow problems [2] actually correspond to cases in which not all involved entities may be controlled. We note that one could also consider flow games among more than two players, and also examine classical algorithmic game-theory questions: existence of a Nash equilibrium, stability inefficiency, and many more (see Section 7). Two-player games are an important and popular first step in studying richer multiplayer settings. Beyond the technical interest (lower bounds apply to the multiplayer setting, and upper bounds are often extendible), the two-player setting captures also an environment composed of different entities. Even when these different entities have no centralized authority, a worst case approach to their joint behavior, which is the one we want to take, amounts to viewing them as a single player.

Other game variants of classical graph-theory problems include the study of *Turán numbers* and *Saturation numbers* for various graph monotone decreasing properties such as triangle freedom, planarity, and 3-colorability. There, two players take turns choosing edges of a given graph as long as the property is preserved in the generated subgraph. One player aims for the process to be long (that is, for the generated subgraph to be big), and the second aims for it to be short [18, 21]. Likewise, the *game chromatic number* of graphs is the smallest number of colors with which a graph can be colored when two players alternate turns in deciding the color of the next vertex [5]. Finally, in *spanning-tree games*, the players alternate turns choosing edges of a weighted graph as long as a cycle is not closed. One player aims to maximize the weight of the generated spanning tree and the second aims to minimize it [22].

The transition from closed to open systems does not come without a computational price. Graph reachability is in NLOGSPACE, while alternating reachability is PTIME-complete. Boolean satisfiability is NP-complete, while quantified Boolean satisfiability is PSPACE-complete. Temporal satisfiability is PSPACE-complete, while temporal synthesis is 2EXPTIME-complete. Understanding the computation cost from going from network flow to network-flow games is a major theme of this work.

We study here *acyclic* game networks. There, given strategies for the players, it is possible to calculate the flow by following a topological ordering of the vertices. We first show that the problem of finding the maximal flow as well as an optimal strategy for the authority in a flow game is Σ_2^P-complete, and is already Σ_2^P-hard to approximate. We point to easy fragments and variants: when the network is a *tree*, and the goal is to maximize the flow that reaches the leaves, and when the environment may *swallow* flow.

Once we find the complexity of flow games, we turn to study three important variants of the setting. Recall that when the incoming flow is larger than the capacity of the outgoing edges, then flow is lost: it gets stuck or leaks. Sometimes it is desirable to find a strategy of Player 0 such that for every strategy of Player 1, there is no loss of flow. Indeed, in some applications the authority cannot tolerate loss of traffic and is willing to reduce the flow in order to ensure no loss. We study the problem of finding the maximal flow that the authority can transfer while ensuring no loss. As good news, we show that the problem of deciding whether some positive flow can be transferred is PTIME-complete, as opposed to the Σ_2^P-completeness in the "with loss" setting. Finding, however, the maximal flow that can be guaranteed with no loss stays Σ_2^P-complete.

---

game theory. This includes, for example, network formation games [4] or incentive issues in interdomain routing and the BGP protocol [13]. We are the first, however, to consider the maximal-flow problem from this perspective.

Our definition of strategies assumes that policies are integral: vertices receive integral incoming flow and partition it to integral flows in the outgoing edges. Integral-flow games arise naturally in settings in which the objects we transfer along the network cannot be partitioned into fractions, as is the case with cars, packets, and more. Sometimes, however, as in the case of liquids, flow can be partitioned arbitrarily. In the traditional maximum-flow problem, it is well known that the maximum flow can be achieved by integral flows [16]. We show that, interestingly, the game setting makes strategies that use non-integral flow stronger: partitioning outgoing flows into non-integers may increase the flow that Player 0 can ensure. Moreover, the gain cannot be bounded by a constant. We leave open the problem of finding the flow that Player 0 can ensure when using non-integral strategies. Hardness in Σ_2^P holds also for this setting. Yet, as real numbers are second-order creatures, the problem may be undecidable.

As a third variant, we define *dynamic flow games*, in which the policy for a vertex u is chosen only when flow reaches it. More formally, the policy in u is chosen once flow in all edges that precede u in the topological order has traveled. We show that the dynamic setting enables a formulation of the problem by means of the first-order theories of integral addition or real addition, making the related decision problems decidable for both integral and non-integral flows [15, 14].

Finally, we discuss additional variants, which we leave for future research: The first is motivated by *evacuation* scenarios [25]: Consider, for example, a road network of a city, where the vertex s denotes the center of the city and the vertex t denotes the area outside of the city. In order to evacuate the center of the city, drivers are asked to navigate from s to t. In each vertex, every incoming driver chooses an arbitrary outgoing edge. If the outgoing capacity from a vertex is smaller than the incoming flow, then a traffic jam occurs, and flow is lost. We want to find the number of cars that are guaranteed to evacuate in the worst scenario. This corresponds to solving a flow-game in which all vertices belong to Player 1.[3] Then, in multiplayer flow games, the vertices of the network are partitioned among several (possibly more than 2) players, each having her target vertex. Finally, in partially-specified flow networks, a strategy is known for some vertices, and we need to find an optimal strategy for the others.

Due to the lack of space, some details and proofs are omitted, and can be found in the full version, in the authors' URLs.

2 Flow Games

A *flow network* is $N = \langle V, E, c, s, t\rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, and $c : E \to \mathbb{N}$ is a capacity function, assigning to each edge an integral amount of flow that the edge can transfer, and $s, t \in V$ are source and target vertices. We assume that t is reachable from s and the capacities are given in unary. A *flow game* between two players, denoted Player 0 and Player 1, is $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$, where V_0 and V_1 are sets of vertices and $\langle V_0 \cup V_1, E, c, s, t\rangle$ is a flow network in which the set of vertices is partitioned between Player 0 and Player 1. Intuitively, Player 0 directs flow that arrives to vertices in V_0 and his goal is to maximize the flow from s to t. Then, Player 1 directs flow that arrives to vertices in V_1 and his goal is to minimize the flow from s to t. A *sink* is a vertex u with no outgoing edges. That is, there is no vertex v with $E(u, v)$. We assume that t is a sink, $s \in V_0$, and that no edge enters s. That is, there is no vertex v with $E(v, s)$.

[3] We note that this is different from work done in *evacuation planning*, where the goal is to find routes and schedules of evacuees (for a survey, see [31]).

Let $V = V_0 \cup V_1$. For a vertex $u \in V$, let E^u and E_u be the sets of incoming and outgoing edges to and from u, respectively. That is, $E^u = (V \times \{u\}) \cap E$ and $E_u = (\{u\} \times V) \cap E$. A *policy* for a vertex $u \in V$, for $u \neq s$, is a function that partitions an incoming flow between the outgoing edges. Formally, a policy for u is a function $f_u : \mathbb{N} \to \mathbb{N}^{E_u}$ such that for every flow $x \in \mathbb{N}$ and edge $e \in E_u$, we have $f_u(x)(e) \leq c(e)$ and $\sum_{e \in E_u} f_u(x)(e) = \min\{x, \sum_{e \in E_u} c(e)\}$. Thus, $f_u(x)$ assigns to each edge outgoing from u a flow that is bounded by its capacity. Also, when the incoming flow is larger than the capacity of the outgoing edges (which bounds the outgoing flow), then flow is *lost* and the outgoing flow is lower than the incoming flow. In practice, loss of flow may correspond to leaks – fluid in a pipe system that is lost when the system is overflowed, to traffic that gets stuck – in jammed road systems, or to packets that are thrown by routers all whose outgoing channels are filled. Note that this is different from the traditional definition of flow in a network, which corresponds to the case all vertices belong to Player 0, and in which the "flow conservation" property is respected. Note also that $f_u(0)(e) = 0$ for every e. For the source vertex s, a policy is a function $f_s \in \mathbb{N}^{E_s}$ such that for every edge $e \in E_s$, we have $f_s(e) \leq c(e)$.

A *flow* in a flow game is a function $f \in \mathbb{N}^E$ that assigns to each edge the flow that travels in it. We require that for every edge $e \in E_u$, we have $f(e) \leq c(e)$, and for every vertex $u \in V$, except for s and t, we have $\sum_{e \in E_u} f(e) = \min\{\sum_{e \in E^u} f(e), \sum_{e \in E_u} c(e)\}$. That is, the flow in each edge is bounded by its capacity, and the flow that leaves each vertex is the minimum between the flow that enters the vertex and the sum of the capacities of edges outgoing from it. We focus on the case where the graph $\langle V, E \rangle$ is acyclic. Then, given policies f_u for all vertices in $u \in V$, we can calculate the flow in the game as follows. First, we order the vertices in a topological ordering. Then, we start from the vertex s (we ignore every vertex preceding s), and use f_s to assign a flow to each edge in E_s. Now, we continue to the next vertex in the topological ordering. Whenever we reach a vertex u, the incoming flow to u, denoted x, has already been calculated. We then use $f_u(x)$ to assign a flow for each edge in E_u, and continue along the topological ordering until we reach t. Since the flow that enters a vertex u depends only on the sub-game that reaches u, it is easy to see that the calculation above is independent of the topological ordering. Indeed, if u_1 and u_2 are not ordered, then flow that leaves u_1 does not reach u_2, and vice versa.

A strategy for Player 0 is a collection of policies, one for each vertex in V_0. Likewise, a strategy for Player 1 is a collection of policies, one for each vertex in V_1. Let F_0 and F_1 be the sets of all possible strategies of Player 0 and Player 1 respectively. Given strategies $\alpha \in F_0$ and $\beta \in F_1$, the flow in the game, denoted $f^{\alpha,\beta}$, can be calculated as described above. The *outcome* of the game with strategies α and β is then the amount of flow that reaches the target t. Thus, $\mathit{outcome}(\alpha, \beta) = \sum_{e \in E^t} f^{\alpha,\beta}(e)$. The *value* of the game, denoted $\mathit{value}(\mathcal{G})$, is defined by $\max_{\alpha \in F_0} \min_{\beta \in F_1} \mathit{outcome}(\alpha, \beta)$. That is, against every strategy of Player 0, we take the most hostile strategy of Player 1, and the value is obtained by considering the strategy of Player 0 that achieves the maximal flow no matter how Player 1 responds. Note that since we quantify on the strategies of Player 1 universally, this corresponds to a setting in which Player 0 proceeds first and determines the policies in all his vertices, and then Player 1 responds by determining the policies in all his vertices.

3 Problem Complexity

In this section we study the complexity of the problem of finding the value of a flow game. We consider the corresponding decision problem, which is parameterized by a threshold. Formally, the input to the *flow-game problem* (FG problem, for short) is a flow game $\mathcal{G}$ and

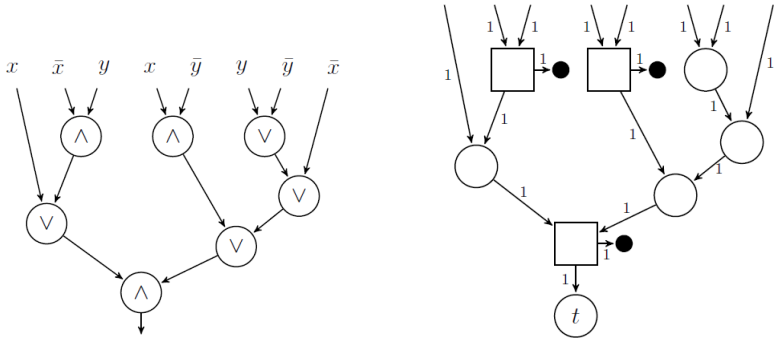


Figure 1 The circuit $\mathcal{C}_\psi$ and the external-source flow game G_ψ, for $\psi = x \vee (\bar{x} \wedge y) \wedge ((x \wedge \bar{y}) \vee (y \vee \bar{y} \vee \bar{x}))$.

a threshold $\gamma \in \mathbb{N}$, and the goal is to decide whether $value(\mathcal{G}) \geq \gamma$.

We show that the problem is complete in Σ_2^P, namely the class of problems that can be solved by a nondeterministic polynomial Turing machine that has an oracle to some NP-complete problem.

▶ **Theorem 2.** *The FG problem is Σ_2^P-complete.*

Proof. We start with the upper bound. Consider a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{N}$. Strategies for Player 0 and Player 1 can be represented by polynomial-length strings. Given a strategy α for Player 0, the problem of checking whether there is a strategy β for Player 1 such that $outcome(\alpha, \beta) < \gamma$ is in NP. Consequently, deciding whether there is a strategy α for Player 0 such that for every strategy β for Player 1 we have $outcome(\alpha, \beta) \geq \gamma$, can be done by a nondeterministic polynomial-time Turing machine with an NP oracle.

We continue to the lower bound and describe a reduction from QBF_2: satisfiability for quantified Boolean formulas with 2 alternations of quantifiers, where the most external quantifier is "exists". Let ψ be a Boolean propositional formula over the variables $x_1, \ldots, x_n, y_1, \ldots, y_m$ and let $\theta = \exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \psi$. Also, let $X = \{x_1, \ldots, x_n\}$, $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$, $Y = \{y_1, \ldots, y_m\}$, $\bar{Y} = \{\bar{y}_1, \ldots, \bar{y}_m\}$, $Z = X \cup Y$, and $\bar{Z} = \bar{X} \cup \bar{Y}$. We construct a flow game $\mathcal{G}_\theta$ such that the value of G_θ is 1 if θ holds and is 0 otherwise.

We assume that ψ is given in a positive normal form, that is, ψ is constructed from the literals in $Z \cup \bar{Z}$ using the Boolean operators $\vee$ and $\wedge$, and that there is $k \geq 1$ such that every literal in $Z \cup \bar{Z}$ appears in ψ exactly k times. Clearly, every Boolean propositional formula can be converted with only a quadratic blow-up to an equivalent one that satisfies these conditions.

We first translate ψ into a Boolean circuit $\mathcal{C}_\psi$ with $k(2n + 2m)$ inputs – one for each occurrence of a literal in ψ. For example, in Figure 1, on the left, we describe C_ψ for $\psi = x \vee (\bar{x} \wedge y) \wedge ((x \wedge \bar{y}) \vee (y \vee \bar{y} \vee \bar{x}))$. Each gate in $\mathcal{C}_\psi$ has fan-in 2 and fan-out 1. We say that an input assignment to $\mathcal{C}_\psi$ is consistent if it corresponds to an assignment to the variables in Z. That is, for each variable $z \in Z$, there is a value $b \in \{0, 1\}$ such that all the k inputs that correspond to the literal z have value b and all the k inputs that correspond to the literal $\bar{z}$ have value $\bar{b}$. If the input to $\mathcal{C}_\psi$ is consistent then $\mathcal{C}_\psi$ computes the value of ψ for the corresponding assignment.

Now, we translate $\mathcal{C}_\psi$ to an *external-source flow game* $\mathcal{G}_\psi = \langle V_0, V_1, E, c, t \rangle$: a flow game in which there is no source vertex, and an input flow is given externally. Formally, some of the edges in E have an unspecified source, to be later connected to edges with an unspecified

target. The idea behind the translation is as follows: The capacities in $\mathcal{G}_\psi$ are all 1. Each OR gate in $\mathcal{C}_\psi$ induces a vertex v_0 in V_0 that has in-degree 2 and out-degree 1. Thus, if the incoming flow in each incoming edge to v_0 is 0 or 1, then its outgoing flow is 1 iff at least one of its incoming edges has flow 1. Then, each AND gate in $\mathcal{C}_\psi$ induces a vertex v_1 in V_1 that has in-degree 2 and out-degree 2, yet, one of the two edges that leaves v_1 leads to a sink. Accordingly, if the incoming flow in each incoming edge to v_1 is 0 or 1, then the outgoing flow in the edges that does not lead to the sink is 1 in all the policies for v_1 iff both incoming edges have flow 1. For example, the Boolean circuit $\mathcal{C}_\psi$ from Figure 1 is translated to the external-source flow game $\mathcal{G}_\psi$ to its right.

The following lemma can be proved by an induction on the structure of ψ.

▶ **Lemma 3.** *Consider a Boolean formula ψ and its corresponding external-source flow game $\mathcal{G}_\psi$.*

1. *Given input flows to $\mathcal{G}_\psi$, if we increase some input flow, then the new value of the game is greater than or equal to the original value.*
2. *Given input flows in $\{0,1\}$ to $\mathcal{G}_\psi$, the value of the game is equal to the output of $\mathcal{C}_\psi$ with the same input. Thus, if the input flows to $\mathcal{G}_\psi$ corresponds to a consistent input to $\mathcal{C}_\psi$, then the value of $\mathcal{G}_\psi$ is the value of ψ for the corresponding assignment.*

We complete the reduction by constructing the flow game $\mathcal{G}_\theta$, which uses $\mathcal{G}_\psi$ as a sub-game. As can be seen in Figure 2, the game $\mathcal{G}_\theta$ has a source s that can direct flow to a layer of *variable vertices* – vertices associated with the variables in Z. The existentially quantified variables, namely these in X, are associated with V_0 vertices, and the universally quantified ones, namely these in Y, are associated with V_1 vertices. The source s can direct $2k$ units of flow to each variable vertex. The third layer of the game consists of *literal vertices*: each variable vertex d_z has two successors, associated with the literals z and $\bar{z}$. Vertices associated with literals in $X \cup \bar{X}$ are in V_1, whereas these associated with literals in $Y \cup \bar{Y}$ in V_0. The edges from a variable vertex to its literal vertices have capacity $2k$. Intuitively, the structure of the game, together with Lemma 3, imply that optimal strategies for both players directs all the $2k$ units that enter a variable vertex d_z into one of its literal vertices, which corresponds to a choice between z and $\bar{z}$. Consequently, there is a correspondence between the outgoing flow from the variable vertices and assignments to the variables, which induces an input to $\mathcal{C}_\psi$. Each outgoing edge from a literal vertex enters an input of $\mathcal{C}_\psi$ that corresponds to this literal.

In the full version we describe the game $\mathcal{G}_\theta$ for the case $\psi = (x \vee y) \wedge (\bar{x} \vee \bar{y})$, and prove that the value of $\mathcal{G}_\theta$ is 1 if θ holds and is 0 otherwise. ◀

Note that from the proof of Theorem 2 we can also conclude that the FG problem is Σ_2^P-hard already for flow games in which the capacities of the edges are 1. Indeed, we can change the reduction so that in the game $\mathcal{G}_\theta$, an edge with capacity d is replaced by d parallel edges with capacity 1.

A natural question that arises is whether we can efficiently find an approximated solution for the FG problem. In particular, we say that a solution x is a δ-approximation for the value of a flow game $\mathcal{G}$ if $\frac{x}{\delta} \leq \mathit{value}(\mathcal{G}) \leq \delta \cdot x$. As we now show, finding an approximated solution to the FG problem is not easier than finding an exact solution. Formally, we have the following.

▶ **Theorem 4.** *It is Σ_2^P-hard to approximate the FG problem within any multiplicative factor.*

Proof. The reduction used for proving the Σ_2^P lower bound in the proof of Theorem 2 is such that the value of the flow game $\mathcal{G}_\theta$ is 1 if θ holds and is 0 otherwise. Thus, deciding

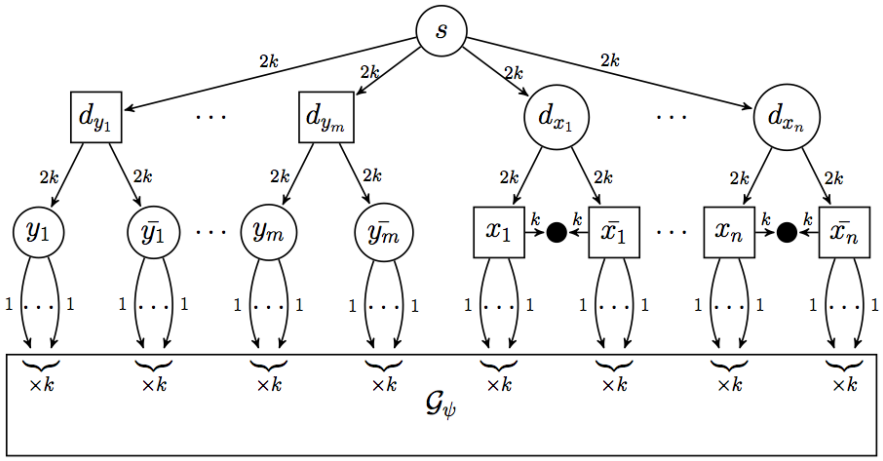


Figure 2 The game $\mathcal{G}_\theta$.

whether the value of a flow game is 0 or 1 is already Σ_2^P-hard. Since every algorithm that approximates the solution of the problems within a multiplicative factor can determine whether the value of the game is positive or is 0, then approximated solution amounts to a precise solution, and is therefore Σ_2^P-hard. ◀

3.1 Feasible Cases

We describe a variant of the problem as well as a structural restriction on games for which the FG problem can be solved efficiently.

Swallowing Flow. A most hostile environment is one that simply swallows all flow that reaches vertices it controls. That is, flow that reaches V_1 is lost. It is easy to analyze flow games in this setting, as they coincide with traditional networks in which the vertices in V_1 are eliminated. Indeed, Player 0 should avoid sending flow to vertices in V_1. It follows that maximal flow in this setting can be found in polynomial time. A variant of this setting is one in which there is a bound on the flow that Player 1 can swallow in each vertex. It is not hard to see that this variant can be reduced to our model by adding an edge to a sink from each vertex of Player 1. The capacity of the edge is the amount of flow that Player 1 can swallow in the vertex.

Tree Flow Games. Let $\mathcal{G} = \langle V_0, V_1, E, c, s\rangle$ be a flow game without a target vertex. Let $V = V_0 \cup V_1$. Assume that the directed graph $\langle V, E\rangle$ is a tree, namely, the in-degree of every vertex in $V \setminus \{s\}$ is 1. The goal of Player 0 is to maximize the flow that enters the leaves of the tree. In the full version we show that if the out-degree of every vertex is bounded by a constant then the FG problem can be solved in polynomial time. Essentially, the value of a game from a vertex u can be expressed as a min-max expression over the possible values of the games that start from the successors of u, which enables the formulation of the problem by means of dynamic programming. When the out-degrees of the vertices are constants, the latter can be solved in polynomial time.

4 No-Loss Flow Games

Recall that when the incoming flow is larger than the capacity of the outgoing edges, then flow is lost and the outgoing flow is lower than the incoming flow. Sometimes it is desirable to find a strategy of Player 0 such that for every strategy of Player 1, there is no loss of flow. We call such a strategy of Player 0 a *no-loss strategy*. Formally, $\alpha \in F_0$ is a no-loss strategy if for every strategy $\beta \in F_1$, there is no vertex $u \in V$ such that the flow that enters u in $f^{\alpha,\beta}$ is larger than the sum of the capacities of the edges that leave u. Thus, for all vertices $u \in V$ except for s and t, we have $\sum_{e \in E_u} f^{\alpha,\beta}(e) = \sum_{e \in E^u} f^{\alpha,\beta}(e)$.

No-loss strategies are required in applications in which the authority cannot tolerate loss of traffic and is willing to reduce the flow in order to ensure no loss. For example, when it involves leaks of hazards or loss of crucial packets in a communication network. The trivial strategy, where the flow is 0, is clearly a no-loss strategy. In this section we study optimal no-loss strategies, namely no-loss strategies that ensure a maximal flow. Given a flow game $\mathcal{G}$, let $nl\_value(\mathcal{G})$ denote the value of the $\mathcal{G}$ when the strategy of Player 0 is restricted to strategies that ensure no loss.

Recall that in Theorem 2 we showed that deciding whether $value(\mathcal{G}) \geq 1$ is Σ_2^P-complete. We start with some good news and show that if Player 0 is restricted to no-loss strategies, the problem can be solved in polynomial time. In order to prove this, we first define *reachability games*.

Consider a graph $\langle V, E\rangle$, source and target vertices $s, t \in V$, and a partition $V_0 \cup V_1$ of V. The reachability game on $\langle V_0, V_1, E, s, t\rangle$ is played between Player 0 and Player 1 as follows. Initially, a token is placed on s. Then, in each step, if the token is placed on a vertex $u \in V_0$ (respectively, $u \in V_1$), then Player 0 (respectively, Player 1) chooses a successor vertex v, namely v such that $\langle u, v\rangle \in E$, and moves the token to v. A strategy of Player 0 (respectively Player 1) maps each vertex $u \in V_0$ (respectively, $u \in V_1$) that is neither t nor a sink to a successor vertex v. A pair of strategies, α for Player 0 and β for Player 1, induces a unique path, denoted $outcome(\alpha, \beta)$, to be traversed by the token when the players follow their strategies. When the graph is acyclic, the path is finite and Player 0 wins if the path reaches t. Otherwise, the path reaches a sink and Player 1 wins. A *winning strategy* for Player 0 is a strategy α such that for every strategy β of Player 1, the path $outcome(\alpha, \beta)$ reaches t. it is well known that the problem of deciding the winner in a reachability game can be solved in linear time and is PTIME-complete [23].

▶ **Theorem 5.** *Consider a flow game $\mathcal{G}$. Deciding whether $nl\_value(\mathcal{G}) \geq 1$ can be done in linear time and is PTIME-complete.*

Proof. We reduce the problem of deciding whether a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$ satisfies the requirement $nl\_value(\mathcal{G}) \geq 1$ to the problem of deciding the reachability game $\mathcal{G}' = \langle V_0, V_1, E, s, t\rangle$, and reduce the reachability game for $\mathcal{G}'$ to the problem of deciding whether a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$ in which all capacities are 1 satisfies $nl\_value(\mathcal{G}) \geq 1$. The upper and lower bounds then follow from known complexity of reachability games. In both reductions, a no-loss strategy for Player 0 in $\mathcal{G}$ is related with a strategy for Player 0 in $\mathcal{G}'$. Essentially, the successor vertex to which the flow of 1 is going to be directed in a no-loss strategy is the successor vertex to which the token is going to be moved, and vice versa. The details of the reductions can be found in the full version. ◀

While deciding whether a positive outcome can be achieved by a no-loss strategy can be done in polynomial time, calculating the no-loss value of the game is not easier than in the

general case. Formally, in the *no-loss flow game problem* (NLFG problem, for short) the input is a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{N}$, and the goal is to decide whether $nl\_value(\mathcal{G}) \geq \gamma$.

▶ **Theorem 6.** *The NLFG problem is Σ_2^P-complete.*

Proof. The upper bound is similar to the case where loss is allowed. Indeed, given a strategy α for Player 0, the problem of checking whether there is a strategy β for Player 1 such that there is loss or that $outcome(\alpha, \beta) < \gamma$ is in NP, which implies membership in Σ_2^P for the NLFG problem.

For the lower bound, we show that the reduction from QBF_2 described in the proof of Theorem 2 can be manipulated to address no-loss strategies. Essentially, this is done by replacing all sinks by a single vertex in which loss is possible, but may be avoided when the formula is satisfiable. The details of the reduction are described in the full version. ◀

5 Non-Integral Flow Games

Recall that the capacities in the network are integral and that a policy for a vertex can assign only integral flows. Integral-flow games arise naturally in settings in which the objects we transfer along the network cannot be partitioned into fractions, as is the case with cars, packets, and more. Moreover, sometimes, as in cases of messages or other information packages, objects can be partitions up to a known granularity. It is easy to see that by multiplying all capacities by factor γ and solving an integer-flow game in the obtained game, we get a solution that involves strategies with fractions of $\frac{1}{\gamma}$ in the original game. Sometimes, however, as in the case of liquids, flow can be partitioned arbitrarily. In the traditional one-player setting, it is known that when the capacities are integral, then there exists an integral maximum flows. In this section we study an extension of the setting and allow strategies to use flows in $\mathbb{R}$. We show that, interestingly, the game setting makes these strategies stronger, in the sense that partitioning outgoing flows into reals may enlarge the value of the game. Moreover, the gain cannot be bounded by a constant.

Consider a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$. Let $\mathbb{R}_+$ denote the set of non-negative real numbers. A *non-integral policy* for a vertex $u \neq s$ is a function $f_u : \mathbb{R}_+ \to \mathbb{R}_+^{E_u}$ such that for every flow $x \in \mathbb{R}_+$ and edge $e \in E_u$, we have $f_u(x)(e) \leq c(e)$ and $\sum_{e\in E_u} f_u(x)(e) = \min\{x, \sum_{e\in E_u} c(e)\}$. Likewise, a non-integral policy f_s for the source s is a function in $\mathbb{R}_+^{E_s}$. Thus, non-integral policies allow non-integral flows. We say that a strategy is a *non-integral strategy* if it contains non-integral policies. We use $ni\_value(\mathcal{G})$ to denote the non-integral value of $\mathcal{G}$, namely the value when the players are allowed to use non-integral strategies. We first show that Player 0 can benefit from using non-integral strategies:

▶ **Theorem 7.** *There is a flow game $\mathcal{G}$ such that an optimal strategy for Player 0 in $\mathcal{G}$ must be non-integral even when Player 1 is restricted to integral strategies. Thus, $ni\_value(\mathcal{G}) > value(\mathcal{G})$.*

Proof. Consider the flow game appearing in Figure 3 on the left. Consider the non-integral strategy α that assigns an outgoing flow of 2 to $\langle s, u\rangle$ and partitions the incoming flow to v_0 equally between $\langle v_0, v_1\rangle$ and $\langle v_0, v_2\rangle$. It is not hard to see that for every strategy β for Player 1, we have that $outcome(\alpha, \beta) \geq 1.5$. Indeed, the sum of the incoming flows to v_1 and v_2 is 2, implying that there is loss of flow in at most one of these vertices. Since the flow from v_0 to both v_1 and v_2 is at most 0.5, then the loss is at most 0.5. On the other hand, if Player 0 is restricted to integral strategies, then an outcome of more than 1 cannot be ensured, as once Player 0 has chosen his strategy and directs 2 units to u, Player 1 can cause

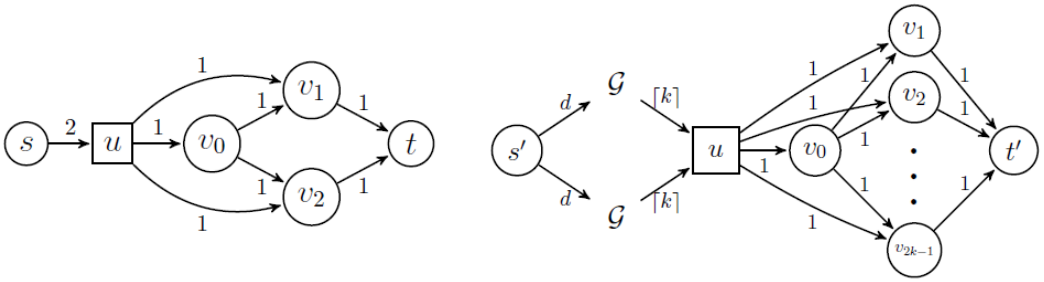


Figure 3 A FG in which Player 0 benefits from non-integral strategies (left) and its amplification (right).

a loss of 1 by choosing a policy in u that directs 1 to v_0 and directs 1 to the vertex (v_1 or v_2) to which Player 0 directs the incoming flow of 1 to v_0. ◀

In the *non-integral flow game problem* (NIFG problem, for short) the input is a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{R}_+$, and the goal is to decide whether $ni\_value(\mathcal{G}) \geq \gamma$. We consider also the *no-loss non-integral flow game problem* (NLNIFG, somewhat shorter), where we consider non-integral strategies that also ensure no loss of flow.

The reduction from QBF_2 described in Theorem 2 holds also for these new settings. Indeed, by Lemma 3, the optimal strategies of the players would result in integral input to $\mathcal{G}_\psi$. Hence, the following theorem.

▶ **Theorem 8.** *The NIFG and NLNIFG problems are Σ_2^P-hard. The NIFG problem is also Σ_2^P-hard to approximate within any multiplicative factor.*

Theorem 8 only gives a lower bound for the problem. The upper bound for the FG problem relied on the fact that integer strategies have a polynomial representation, which cannot be applied in the NIFG problem. Moreover, since reals are second-order creatures, the problem may be undecidable. We leave the decidability problem open and describe, in Section 6, a decidable variant. A natural question is whether an algorithm that solves the FG problem (for integral strategies) approximates the solution for the case of non-integral strategies. As we show in the following theorem, such an approximation cannot be bounded by a constant factor.

▶ **Theorem 9.** *For all $n > 1$, there is a flow game $\mathcal{G}_n$ s.t. $value(\mathcal{G}_n) = 1$ and $ni\_value(\mathcal{G}_n) \geq n$.*

Proof. We amplify the construction used in the proof of Theorem 7. Recall that there, we showed that the flow game in Figure 3 on the left has value 1 and has non-integral value 1.5. The amplification involves two ideas: we parameterize the branching degree of v_0 and we use a recursion that involves sub-games for which a smaller ratio between the value and the non-integral value is known.

Let $\mathcal{G}$ be a flow game such that $value(\mathcal{G}) = 1$ and $ni\_value(\mathcal{G}) = k$, for $k > 1$, and assume that $2k$ is an integer. For example, in the game presented in Figure 3 on the left, we have $k = 1.5$. We construct a flow game $\mathcal{G}'$ such that $value(\mathcal{G}') = 1$ and $ni\_value(\mathcal{G}') = 2k - 1$. By repeating the construction logarithmically many times, we obtain the required flow game $\mathcal{G}_n$.

Consider the game $\mathcal{G}$. Let d_1 and d_2 be the outgoing flow from the source in an optimal strategy and a non-integral strategy of Player 0 in $\mathcal{G}$, respectively. Let d be an integer greater

than d_1 and d_2. We define the new game $\mathcal{G}'$ as shown in Figure 3 on the right. The game $\mathcal{G}'$ contains $\mathcal{G}$ twice as a sub-game: the incoming edge to $\mathcal{G}$ enters its source and the outgoing edge from $\mathcal{G}$ leaves its target. In a flow game on $\mathcal{G}'$, the maximum flow to the vertex u that Player 0 can ensure with an integral strategy is 2. Hence, $value(\mathcal{G}') = 1$. Allowing non-integral strategies, the maximum flow to the vertex u that Player 0 can ensure is $2k$, making $ni\_value(\mathcal{G}') = 2k - 1$. ◀

6 Dynamic Flow Games

In our definition of flow games, the players choose their strategies before the game starts. The universal quantification on the strategies of Player 1 implies that this is equivalent to a scenario in which Player 0 chooses a strategy and then Player 1 chooses a strategy. In dynamic flow games we refer to the way information flows in the graph and let the players choose policies in a vertex only after flow travels in the subgraph from which the vertex is reachable. Since the graph is acyclic, this is well defined.

Formally, we first order the graph in a topological ordering such that V_0 vertices appear, whenever possible, before V_1 vertices. That is, we fix a topological ordering $\leq$ such that if there is neither a path from $u \in V_0$ to $v \in V_1$ nor from v to u then $u \leq v$. Now, Player 0 chooses a flow for each outgoing edge of s. Then, we follow the order $\leq$ and for each vertex u, the player that controls u directs the incoming flow to the outgoing edges. Thus, instead of choosing a strategy that describes the behavior for each possible incoming flow for each vertex u, the player just chooses how to direct the incoming flow, and he has information on the flow in edges whose source precedes u in $\leq$. The value of the game is the maximal flow to t that Player 0 can ensure.

Note that the dynamic setting is not equivalent to the non-dynamic (original) one. For example, the value of the flow game in Figure 3 is 1 (and is 1.5 when we allow non-integral strategies), whereas the dynamic setting leads to a value of 2. The flow game in Figure 3 suggests that the information about "earlier" flow helps Player 0 to use integral flows. Indeed, coming to decide the flow from vertex v_0, Player 0 already known which of v_1 and v_2 have already received 1 flow unit. As we show in the full version, however, the information may not be sufficient in general, and Player 0 can benefit from using non-integral strategies.

The Σ_2^P lower bounds described in Sections 3 and 4 apply also for the dynamic setting. On the positive side, the dynamic setting enables a formulation of the problem by means of the first-order theories of integral addition or real addition, making the related decision problems decidable for both integral and real flows [15, 14].

7 Discussion and Future Work

Today's computing environment involves parties that should be considered adversarial. This calls for a re-examination of classical algorithmic problems. We introduced and studied flow games, which capture settings in which the authority can control the flow only in part of the vertices in a flow network. Below we discuss possible extensions of our work as well as open problems we have left for future research.

7.1 Extensions

Evacuation. An evacuation flow game is $\mathcal{G} = \langle \{s\}, V_1, E, c, s, t \rangle$. Thus, all vertices except for the source belong to Player 1.[4] As explained in Section 1, an evacuation flow game corresponds to scenarios in which the authority has no control on how flow travels on the network [25]. Solving the FG problem in an evacuation flow game amounts to finding a strategy for Player 1 that minimizes the flow.

Beyond the clear practical applications, it is interesting to study the computational aspects of the evacuation problem. As it refers only to one player, and thus involves no alternation of quantification, there is hope the problem can be solved in polynomial time, possibly by a reduction to a prioritized variant of the maximal-flow problem. The latter is strongly related to several weighted variants of the max-flow problem. In particular, polynomial solutions are known for the min-cost max-flow problem [34] and to max-flow with lexicographically ordered edges problem [26].

A Classical Algorithmic Game-Theory Examination. Beyond the questions we studied for FGs, the game setting calls for a classical algorithmic game-theory examination: existence of a Nash equilibrium, stability inefficiency, and many more. We plan to study them for *multiplayer flow games*, where the vertices of the network are partitioned between some $k > 1$ players, there are k target vertices, and the objective of each player is to direct as much flow as possible to her target vertex. For example, in a communication network controlled by several companies (that is, vertices correspond to routers owned by the companies), each company wants to maximize the number of messages from its source to target vertices, and we want to find the best Nash equilibria or design networks for which this equilibria is close to the social optimum.

Partially Specified Networks. In some cases, a strategy is known for some vertices and we need to find an optimal strategy for the others. For example, in hybrid networks, where some vertices are new SDN routers whose behaviour can be controlled, and the other vertices run known traditional routing protocols (e.g., route via the shortest path) [3]. As another example, consider the case where each time that a driver reaches an uncontrolled vertex, it randomly chooses an outgoing edge with free capacity. In this case, we need to find a strategy that maximizes the expected flow to the target.

7.2 Open Problems

Recall that a maximum flow in a flow game may involve non-integrals even when all capacities are integrals. While we settle the complexity of flow games for strategies that only use integrals, decidability for the setting in which strategies may use non-integrals is open. One approach to obtain decidability and analyze flow games with non-integral strategies is to approximate their value with integral strategies. While such an approximation cannot be bounded by a constant multiplicative factor (Theorem 9), we conjecture that it can be bounded by a linear additive factor. In particular, we conjecture that for every flow game $\mathcal{G}$, we have that $ni\_value(\mathcal{G}_n) \leq value(\mathcal{G}_n) + |E|$, where the intuition is that a non-integral

[4] Recall that we define a flow game with $s \in V_0$. Alternatively, we could have defined an evacuation flow game as one in which all vertices are in V_1 and parameterize the game with an initial flow that enters s. Indeed, saturating the edges from s is a dominating strategy for Player 0, so Player 0 does not really take decisions even when $s \in V_0$.

flow in $\mathcal{G}$ should not exceed flow in the game obtained from $\mathcal{G}$ by increasing all capacities by 1. Another approach would be to find a granularity to which it would be sufficient to break an integral flow. Our efforts in this direction so far reveal some counter-intuitive facts. For example, flow games are not *local monotonic*: increasing the flow that enters a vertex may cause a decrease in the flow directed by an optimal strategy to one of its outgoing edges. Also, the required granularity is not continuous, in the following sense: for every $k \geq 1$, we can generate a FG such that an optimal strategy that can break the flow into the fractions $\frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k-1}$ achieves the same flow achieved by an optimal strategy that only uses integrals, yet breaking the flow into the fraction $\frac{1}{k}$ improves this flow. Also, while we know that non-integral strategies may be better than integral ones, we do not know whether strategies that use rational numbers are as good as strategies that use real ones.

Acknowledgment. We thank David Hay for helpful discussions on software-defined networks.

References

1. S. Agarwal, M. S. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *Proc. 32nd IEEE International Conference on Computer Communications*, pages 2211–2219, 2013.
2. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall Englewood Cliffs, 1993.
3. B. Aminof, O. Kupferman, and R. Lampert. Rigorous approximated determinization of weighted automata. *Theoretical Computer Science*, 480:104–117, 2013.
4. E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.
5. T. Bartnicki, J. A. Grytczuk, H. A. Kierstead, and X. Zhu. The map coloring game. *AmericanMathematical Monthly*, 114:793–803, 2007.
6. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.
7. S.A. Cook. Path systems and language recognition. In *Proc. 2nd ACM Symp. on Theory of Computing*, pages 70–72, 1970.
8. S.A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.
9. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.
10. E.A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doll*, 11(5):1277–1280, 1970. English translation by RF. Rinehart.
11. J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.
12. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.
13. J. Feigenbaum, C.H. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. *Distributed Computing*, 18(1):61–72, 2005.
14. J. Ferrante and C. Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.
15. M.J. Fischer and M.O. Rabin. Super-exponential complexity of presburger arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 27–41, 1974.

16 L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

17 L.R. Ford and D.R. Fulkerson. *Flows in networks*. Princeton Univ. Press, Princeton, 1962.

18 Z. Füredi, D. Reimer, and A. Seress. Triangle-free game and extremal graph problems. *Congressus Numerantium*, 82:123–128, 1991.

19 A.V. Goldberg, É. Tardos, and R.E. Tarjan. Network flow algorithms. Technical report, DTIC Document, 1989.

20 A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

21 D. Hefetz, M. Krivelevich, A. Naor, and M. Stojaković. On saturation games. *European Journal of Combinatorics*, 41:315–335, 2016.

22 D. Hefetz, O. Kupferman, A. Lellouche, and G. Vardi. Spanning-tree games. To appear, 2018.

23 N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and Systems Science*, 22(3):384–406, 1981.

24 E. Kalai and E. Zemel. Totally balanced games and games of flow. *Mathematics of Operations Research*, 7(3):476–478, 1982.

25 S. Keren, A. Gal, and E. Karpas. Goal recognition design for non optimal agents. In *Proc. 29th AAAI conference*, pages 3298–3304, 2015.

26 D. Kozen. Lexicographic flow. Technical Report http://hdl.handle.net/1813/13018, Computing and Information Science, Cornell University, June 2009.

27 O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

28 N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

29 M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

30 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

31 L. Qingsong, G. Betsy, and S. Shashi. Capacity constrained routing algorithms for evacuation planning: A summary of results. In *International Symposium on Spatial and Temporal Databases*, pages 291–307. Springer, 2005.

32 B.L. Schwartz. Possible winners in partially completed tournaments. *SIAM Review*, 8(3):302–308, 1966.

33 L.J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.

34 É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

35 S. Vissicchio, L. Vanbever, and O. Bonaventure. Opportunities and research challenges of hybrid software defined networks. *Computer Communication Review*, 44(2):70–75, 2014.

A Dichotomy Theorem for the Inverse Satisfiability Problem

Victor Lagerkvist[*1] **and Biman Roy**[†2]

1 **Institut für Algebra, TU Dresden, Dresden, Germany**
victor.lagerqvist@tu-dresden.de
2 **Department of Computer and Information Science, Linköping University, Linköping, Sweden**
biman.roy@liu.se

Abstract

The *inverse satisfiability problem* over a set of Boolean relations Γ (Inv-SAT(Γ)) is the computational decision problem of, given a relation R, deciding whether there exists a SAT(Γ) instance with R as its set of models. This problem is co-NP-complete in general and a dichotomy theorem for finite Γ containing the constant Boolean relations was obtained by Kavvadias and Sideri. In this paper we remove the latter condition and prove that Inv-SAT(Γ) is always either tractable or co-NP-complete for all finite sets of relations Γ, thus solving a problem open since 1998. Very few of the techniques used by Kavvadias and Sideri are applicable and we have to turn to more recently developed algebraic approaches based on *partial polymorphisms*. We also consider the case when Γ is infinite, where the situation differs markedly from the case of SAT. More precisely, we show that there exists infinite Γ such that Inv-SAT(Γ) is tractable even though there exists finite $\Delta \subset \Gamma$ such that Inv-SAT(Δ) is co-NP-complete.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.0 Discrete Mathematics General

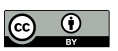
Keywords and phrases Clone Theory, Universal Algebra, Satisfiability Problems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.39

1 Introduction

A *constraint language* is a set of Boolean relations. The *parameterized satisfiability problem* over a constraint language Γ (SAT(Γ)) is the computational decision problem of determining whether a conjunctive formula over Γ is satisfiable. In a seminal paper by Schaefer it was proven that SAT(Γ) is either always tractable or is NP-complete [21]; a property that should not be taken for granted in light of the NP-intermediate problems constructed by Ladner [15]. In this paper we will study the computational complexity of the *inverse satisfiability problem* over a constraint language Γ (Inv-SAT(Γ)), which, as the name suggests, is the exact opposite of SAT(Γ). Hence, instead of a SAT(Γ) instance, we are given a relation R, and the question is then to determine if there exists an instance of SAT(Γ) with precisely R as it sets of models. In fact, for every problem in NP there exists a corresponding inverse problem, and we refer the reader to Chen [7] for a survey on this topic. Contrary to SAT(Γ), Inv-SAT(Γ)

* The author has received funding from the DFG-funded project "Homogene Strukturen, Bedingungserfüllungsprobleme, und topologische Klone" (Project number 622397).
† The author is partially supported by the *National Graduate School in Computer Science* (CUGS), Sweden.



is in general co-NP-complete, and its computational complexity was studied by Kavvadias and Sideri [14]. While a complete dichotomy theorem was not obtained, Kavvadias and Sideri proved that for finite constraint languages Γ containing the constant relations $\{(0)\}$ and $\{(1)\}$, Inv-SAT(Γ) is always either tractable or co-NP-complete. We will strengthen this result and give a complete dichotomy theorem for Inv-SAT(Γ) for finite constraint languages, and thus solve a long-standing open problem. At a first glance, the condition that Γ contains the constant relations might only look like a minor technical difficulty, but there are several reasons why Inv-SAT(Γ) has previously escaped a complete complexity classification. First, for SAT and its multi-valued generalization CSP, it is known that the introduction of constant relations does not affect the complexity of the problem, provided that the constraint language satisfies the algebraic property of being a core. Such a property does not hold a priori for Inv-SAT(Γ), which increases the number of cases we need to consider. Second, and perhaps most importantly, the majority of dichotomies for CSP and for Boolean problems parameterized by constraint languages, have been obtained via the so-called *algebraic approach*. For a thorough survey of this approach we refer the reader to Creignou et al. [8] and to Barto [1]. In short, the algebraic approach allows us to relate the complexity of a problem parameterized by a set of relations Γ to properties of the *polymorphisms* of Γ, which we may think of as a collection of functions preserving the structure of the relations in Γ. The main applicability of this connection is that sets of polymorphisms are well-studied and are in fact completely determined in the Boolean domain [19]. Hence, instead of directly reasoning by properties of constraint languages, we can instead prove complexity results by exploiting properties of well-known polymorphisms. The Inv-SAT(Γ) problem, however, is fundamentally incompatible with polymorphisms, and instead we turn to the more refined concept of *partial polymorphisms*. Unfortunately, partial polymorphisms are not nearly as well-studied as total polymorphisms, which makes such classifications more problematic. To tackle this issue we use the algebraic techniques developed by Schnoor and Schnoor [23] and Lagerkvist [16] and are able to classify the constraint languages under consideration according to their expressive power, in an extremely fine-grained way. These expressibility results turn out to be vital when we prove our dichotomy theorem for Inv-SAT(Γ) in Section 3. More precisely, our dichotomy result states that Inv-SAT(Γ) is co-NP-complete for finite Γ if the polymorphisms of Γ can be generated by a set of unary Boolean operations — a property which in the literature is also sometimes called *non-Schaefer*. This complexity classification in fact exactly coincides with the complexity of enumerating the solutions of SAT(Γ) with polynomial delay [10].

After having proven the dichotomy theorem for Inv-SAT(Γ) for finite Γ we investigate the case when Γ is infinite in Section 4. For SAT(Γ), Schaefer's dichotomy theorem remain valid also for infinite languages, and given the similarity between SAT(Γ) and Inv-SAT(Γ), one might conjecture that the same is also true for Inv-SAT(Γ). Somewhat surprisingly, this turns out to be false: we show that there exists an infinite constraint language Γ such that (1) Inv-SAT(Γ) is tractable, (2) SAT(Γ) is NP-hard, and (3) there exists finite $\Delta \subset \Gamma$ such that Inv-SAT(Δ) is co-NP-complete. Hence, for infinite languages, the complexity of Inv-SAT(Γ) is markedly different from the complexity of enumeration, even though the complexity coincides for finite languages. Moreover, we provide an algebraic criterion for this phenomena based on the expressive power of the partial polymorphisms of Γ, and conjecture that this property is both necessary and sufficient.

2 Preliminaries

A *Boolean relation* is a subset of $\{0,1\}^n$ for some $n \geq 1$, and if R is a relation we write $\mathrm{ar}(R)$ to denote its arity. For a tuple $t = (x_1, \ldots, x_n)$ we write $t[i]$ to denote the ith element x_i,

and $\Pr_{i_1,\ldots,i_{n'}}(t) = (t[i_1],\ldots,t[i_{n'}])$ to denote the *projection* on the coordinates $i_1,\ldots,i_{n'} \in \{1,\ldots,n\}$. Similarly, for an n-ary relation R we let $\Pr_{i_1,\ldots,i_{n'}}(R) = \{\Pr_{i_1,\ldots,i_{n'}}(t) \mid t \in R\}$. We will typically use first-order logical formulas to define relations, and write $R(x_1,\ldots,x_n) \equiv \varphi(x_1,\ldots,x_n)$ to define the relation $R = \{(f(x_1),\ldots,f(x_n)) \mid f \text{ is a model of } \varphi(x_1,\ldots,x_n)\}$. Let BR denote the set of all Boolean relations and $\Pi_{\mathbb{B}}$ the set of all Boolean projections, i.e., operations of the form $\pi_i^n(x_1,\ldots,x_i,\ldots,x_n) = x_i$. A (not necessarily finite) $\Gamma \subseteq BR$ is called a *Boolean constraint language*, or, if there is no risk for confusion, simply a constraint language. If $\{(0)\},\{(1)\} \in \Gamma$ then we say that Γ is *ultraidempotent*. We prefer the term ultraidempotent over idempotent since the latter typically only requires that the constant relations are primitively positively definable (see Section 2.2 for a definition of this concept).

2.1 The Inverse Satisfiability Problem

The *parameterized satisfiability problem* over a constraint language Γ (SAT(Γ)) is the computational decision problem defined as follows.

> INSTANCE: A tuple (V,C) where V is a set of variables and C a set of constraint applications of the form $R(x_1,\ldots,x_{\mathrm{ar}(R)})$ where $R \in \Gamma$ and $x_1,\ldots,x_{\mathrm{ar}(R)} \in V$.
> QUESTION: Does there exist a function $f : V \to \{0,1\}$ such that $(f(x_1),\ldots,x_{\mathrm{ar}(R)}) \in R$ for every $R(x_1,\ldots,x_{\mathrm{ar}(R)}) \in C$?

▶ **Example 1.** Let $R_{1/3}$ be the ternary relation $\{(0,0,1),(0,1,0),(1,0,0)\}$. Then SAT($\{R_{1/3}\}$) can be viewed as a formulation of the 1-in-3-SAT problem without negation, and is well-known to be NP-complete.

We will sometimes view a SAT(Γ) instance as a conjunctive formula φ and write $Sols(\varphi)$ to denote its set of models. The *inverse satisfiability problem* over a constraint language Γ (INV-SAT(Γ)) can then be viewed as the problem of, given a relation R, determining whether there exists a SAT(Γ) instance with precisely R as it set of models. More formally, we define INV-SAT(Γ) as follows.

> INSTANCE: A Boolean relation R.
> QUESTION: Does there exist a SAT(Γ) instance φ such that $Sols(\varphi) = R$?

If this question can be answered in polynomial time with respect to the number of bits required to represent R then we say that INV-SAT(Γ) is tractable. In general the INV-SAT(Γ) problem is co-NP-complete and a dichotomy theorem is known for finite and ultraidempotent constraint languages Γ [14].

▶ **Theorem 2.** *Let Γ be a finite and ultraidempotent constraint language. Then* INV-SAT(Γ) *is either co-NP-complete or tractable.*

▶ **Example 3.** Consider the relation $R_{1/3}$ from Example 1. Then INV-SAT($\{R_{1/3}\}$) is the problem of, given a relation R, deciding if there exists a 1-in-3-SAT instance without negation with exactly R as its set of models. Since $\{R_{1/3}\}$ is not ultraidempotent we cannot however use Theorem 2 to conclude that INV-SAT($\{R_{1/3}\}$) is co-NP-complete. We will return to this problem in Section 3 where we prove our dichotomy theorem for INV-SAT(Γ).

2.2 Closure Operators on Relations

In this section we introduce two closure operators on relations that will be important when explaining the algebraic approach in the forthcoming section. First, if R is an n-ary Boolean relation and Γ a constraint language we say that R has a *primitive positive definition* over Γ if $R(x_1, \ldots, x_n) \equiv \exists y_1, \ldots, y_{n'}.R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$, where each $R_i \in \Gamma \cup \{(0,0),(1,1)\}$ and each $\mathbf{x}_i$ is a tuple of variables over $x_1, \ldots, x_n, y_1, \ldots, y_{n'}$ of length $\mathrm{ar}(R_i)$. In other words R is definable over Γ by a (possibly) existentially quantified, conjunctive formula of constraints over Γ and the equality relation $\{(0,0),(1,1)\}$. Given a constraint language Γ we now write $\langle\Gamma\rangle$ to denote the smallest set of relations containing Γ and which is closed under taking pp-definition. Sets of the form $\langle\Gamma\rangle$ are called *relational clones* or *co-clones*.

Similarly, say that an n-ary Boolean relation has a *quantifier-free primitive positive definition* (qfpp-definition) over a constraint language Γ if $R(x_1, \ldots, x_n) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$, where each $R_i \in \Gamma \cup \{(0,0),(1,1)\}$ and each $\mathbf{x}_i$ is a tuple of variables over $x_1, \ldots, x_n$ of length $\mathrm{ar}(R_i)$. Let $\langle\Gamma\rangle_{\not\exists}$ denote the smallest set of relations containing Γ and which is closed under taking qfpp-definitions. These sets are usually called *weak systems* or *weak partial co-clones*. We remark that there is a very strong connection between Inv-SAT(Γ) and the set $\langle\Gamma\rangle_{\not\exists}$. To see this, note that an instance of Inv-SAT(Γ) is simply a relation R, and the question of whether there exists an instance φ of SAT(Γ) with $Sols(\varphi) = R$, can be rephrased as whether R admits a qfpp-definition over Γ, i.e., $R \in \langle\Gamma\rangle_{\not\exists}$. Whenever convenient we will therefore assume that Inv-SAT(Γ) is the problem of checking whether $R \in \langle\Gamma\rangle_{\not\exists}$. We remark that the related problem of checking whether R admits a pp-definition over Γ is tractable for Boolean Γ [9] but co-NEXPTIME-hard for sufficiently large, but finite, domains [24].

2.3 Closure Operators on Operations

Let $f : \{0,1\}^k \to \{0,1\}$ be a k-ary Boolean operation and R an n-ary Boolean relation. We say that f *preserves* R, that f is a *polymorphism* of R, or that R is *invariant* under f, if $f(t_1, \ldots, t_k) \in R$ for every sequence of tuples $t_1, \ldots, t_k \in R$, where

$$f(t_1, \ldots, t_k) = (f(t_1[1], \ldots, t_k[1]), \ldots, (f(t_1[n], \ldots, t_k[n]))).$$

We write Pol(R) for the set of polymorphisms of the relation R and if Γ is a constraint language we let $\mathrm{Pol}(\Gamma) = \bigcap_{\mathrm{R}\in\Gamma} \mathrm{Pol(R)}$. Sets of the form Pol($\Gamma$) are usually called *clones* and are known to be sets of operations containing all projections (i.e., $\Pi_{\mathbb{B}} \subseteq \mathrm{Pol}(\Gamma)$) and closed under composition (i.e., if $f, g_1, \ldots, g_m \in \mathrm{Pol}(\Gamma)$ where f has arity m and each g_i arity n then the n-ary operation $f \star g_1, \ldots, g_m(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$ is included in Pol(Γ)). We let $[F]$ be the smallest clone containing the set F. There is a powerful connection between clones and co-clones which we will now describe. First, if we let Inv(F) be the set of all relations invariant under the set of operations F, it is known that Inv(F) is in fact closed under pp-definitions, i.e., is a co-clone. Second, for any constraint language Γ it is known that $\mathrm{Inv}(\mathrm{Pol}(\Gamma)) = \langle\Gamma\rangle$, and that for any set of operations F, $\mathrm{Pol}(\mathrm{Inv(F)}) = \mathrm{[F]}$. We now have the following *Galois connection* between Inv($\cdot$) and Pol($\cdot$).

▶ **Theorem 4** ([3, 4, 11]). *Let Γ and Γ' be two constraint languages. Then $\Gamma \subseteq \langle\Gamma'\rangle$ if and only if $\mathrm{Pol}(\Gamma') \subseteq \mathrm{Pol}(\Gamma)$.*

There is a similar Galois connection between weak systems and sets of *partial operations*. Formally, we view a (Boolean) partial operation f of arity k as a mapping $X \to \{0,1\}$ where $X \subseteq \{0,1\}^k$ is called the *domain* of f and denoted by $\mathrm{domain}(f)$. We now say

that a k-ary partial operation f is a *partial polymorphism* of an n-ary relation R if either $f(t_1, \ldots, t_k) \in R$ or there exists $1 \leq i \leq n$ such that $(t_1[i], \ldots, t_k[i]) \notin \text{domain}(f)$, for every sequence $t_1, \ldots, t_k \in R$. We write pPol(R) for the set of all partial polymorphisms of R and pPol(Γ) for the set $\bigcap_{R \in \Gamma} \text{pPol}(\Gamma)$. These sets are usually referred to as *strong partial clones* and are known to be sets of partial operations containing all projections, closed under composition, and closed under taking subfunctions. More precisely, composition of partial operations is defined in exactly the same way as composition of total operations, but the resulting partial operation is only defined for a sequence of arguments if every partial operation in the composition is defined; and by closed under taking subfunctions we mean that if $f \in \text{pPol}(\Gamma)$ then $g \in \text{pPol}(\Gamma)$ for every g such that $\text{domain}(g) \subseteq \text{domain}(f)$ and g matches the values of f for these arguments. We write $[F]_s$ for the smallest strong partial clone containing F, and say that $[F]_s$ is *finitely generated* if there exists finite $G \subseteq [F]_s$ such that $[F]_s = [G]_s$, is infinitely generated otherwise, and in both cases we say that G is a *base* of $[F]_s$. The reason why we define these technical concepts will be explained in Section 4 where we study the complexity of Inv-SAT(Γ) when pPol(Γ) is finitely generated.

Similar to the total case, if we let Inv(F) be the set of relations invariant under the set of partial operations F, then it is known that Inv(F) is closed under qfpp-definitions, and is therefore a weak system. Moreover, $\langle\Gamma\rangle_{\not\exists} = \text{Inv}(\text{pPol}(\Gamma))$ and $[F]_s = \text{pPol}(\text{Inv}(\text{F}))$. We then have the following Galois connection between Inv(·) and pPol(·), due to Geiger [11] and Romov [20].

▶ **Theorem 5** ([11, 20]). *Let* Γ *and* Γ′ *be two constraint languages. Then* $\Gamma \subseteq \langle\Gamma'\rangle_{\not\exists}$ *if and only if* $\text{pPol}(\Gamma') \subseteq \text{pPol}(\Gamma)$.

Using the results in this section we can now present the dichotomy theorem from Kavvadias and Sideri [14] more precisely as follows.

▶ **Theorem 6.** *Let* Γ *be a finite and ultraidempotent constraint language. Then* Inv-SAT(Γ) *is co-NP-complete if* $\text{Pol}(\Gamma) = \Pi_{\mathbb{B}}$ *and is tractable otherwise.*

We remark that the tractable cases in Theorem 6 stem from the observation that if one can enumerate the solutions of SAT(Γ) with polynomial delay, then Inv-SAT(Γ) must be tractable. To see this, let R be an instance of Inv-SAT(Γ), and begin by computing a qfpp-definition φ over Γ with the property that $Sols(\varphi) \supseteq R$, according to the strategy in Kavvadias and Sideri [14]. Then it is sufficient to enumerate at most $|R| + 1$ solutions to φ (viewed as an instance of SAT(Γ)) and stop if any of these solutions do not match the tuples in R.

3 A Dichotomy Theorem for Inv-SAT(Γ)

In this section we will extend Theorem 6 to finite constraint languages that are not necessarily ultraidempotent, in order to obtain a complete dichotomy theorem for Inv-SAT(Γ). First observe that the tractable cases of Theorem 6 remain valid even if Γ is not ultraidempotent since the enumeration algorithms works equivalently well in these cases. To better describe the remaining cases we will need to define the following Boolean operations.

▶ **Definition 7.** We define the following Boolean operations.

1. $f_0(x) = 0$,
2. $f_1(x) = 1$,
3. $\overline{x} = 1 - x$

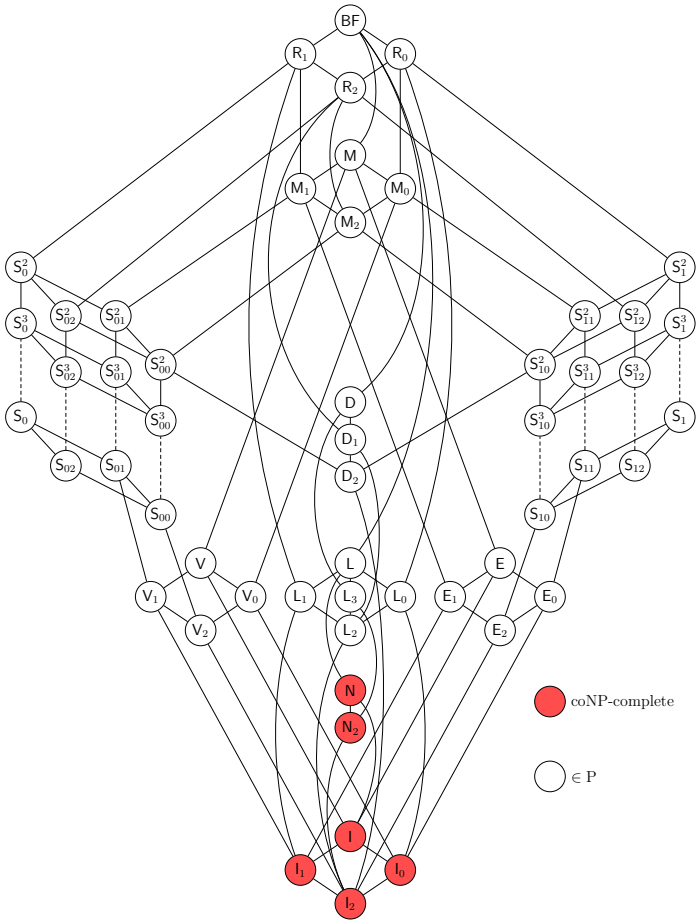


Figure 1 The complexity of Inv-SAT(Γ) for finite Γ.

Then, using the terminology from Böhler et al. [5, 6], $[\{f_0, f_1, \overline{x}\}] = \mathsf{N}$, $[\{f_0, f_1\}] = \mathsf{I}$, $[\{f_0\}] = \mathsf{I}_0$, $[\{f_1\}] = \mathsf{I}_1$, $[\{\overline{x}\}] = \mathsf{N}_2$, and $[\{\pi_1^1\}] = \mathsf{I}_2 = \Pi_{\mathbb{B}}$. Our aim is now to prove the following theorem, which is visualized in Figure 1. The intuition behind the theorem is that one cannot enumerate the solutions of SAT(Γ) with polynomial delay if Pol(Γ) ⊆ [F] for $F \subseteq \{f_0, f_1, \overline{x}\}$, unless P = NP [10].

▶ **Theorem 8.** *Let* Γ *be a finite constraint language. Then* Inv-SAT(Γ) *is co-NP-complete if* Pol(Γ) ⊆ [F] *for* $F \subseteq \{f_0, f_1, \overline{x}\}$ *and is tractable otherwise.*

The theorem will be proved in Lemma 11, Lemma 12, Lemma 14, and Lemma 16. At this stage it might be helpful to review how dichotomy theorems for problems parameterized by Boolean constraint languages are usually obtained. Hence, assume that $X(\Gamma)$ is a computational decision problem for which it is true that $X(\Gamma)$ admits a polynomial-time reduction to $X(\Delta)$ whenever Pol(Δ) ⊆ Pol(Γ). Then, what one needs to do is simply to take every clone Pol(Γ) in Post's lattice and determine the complexity of $X(\Gamma)$, since the results then automatically carry over to every $X(\Delta)$ such that Pol(Δ) = Pol(Γ). This is e.g. the case for SAT and many Boolean optimization and logical reasoning problems [8]. For the Inv-SAT(Γ) problem we do not have such a result, implying that the proof strategy is more complex. However, we will see that it is possible to overcome this using properties of weak systems. For this we will need the following lemma.

▶ **Lemma 9.** *Let* $\mathrm{Pol}(\Gamma) \subseteq [\mathrm{F}]$ *for* $F \subseteq \{f_0, f_1, \bar{x}\}$. *Then*

1. $\tau^{01} = \{(0,1)\}, \tau^{01}_{\neq} = \{(0,1,0,1),(1,0,0,1)\} \in \langle\Gamma\rangle_{\nexists}$ *if* $\mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$,
2. $\tau_{\neq} = \{(0,1),(1,0)\} \in \langle\Gamma\rangle_{\nexists}$ *if* $\mathrm{Pol}(\Gamma) = [\{\bar{\mathrm{x}}\}]$,
3. $\tau_{f_0,f_1,\bar{x}} = \{(0,0,0,0),(1,1,1,1),(0,1,0,1),(1,0,0,1),(1,0,1,0),(0,1,1,0)\} \in \langle\Gamma\rangle_{\nexists}$ *if* $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0, \mathrm{f}_1, \bar{\mathrm{x}}\}]$,
4. $\tau_{\to} = \{(0,0),(1,0),(1,1)\} \in \langle\Gamma\rangle_{\nexists}$ *if* $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0, \mathrm{f}_1\}]$,
5. $\tau^{01}_{\neq} \cup \{(0,0,0,0)\} = \{(0,0,0,0),(0,1,0,1),(1,0,0,1)\} \in \langle\Gamma\rangle_{\nexists}$ *if* $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0\}]$, *and*
6. $\tau^{01}_{\neq} \cup \{(1,1,1,1)\} = \{(0,1,0,1),(1,0,0,1),(1,1,1,1)\} \in \langle\Gamma\rangle_{\nexists}$ *if* $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_1\}]$.

Proof. We consider each case in turn. The various cases follow a similar structure and make use of the algebraic machinery developed by Schnoor and Schnoor [23] and Lagerkvist [16]. We first remark that for $\mathrm{Pol}(\Gamma) \in \{[\{\mathrm{f}_0\}], [\{\mathrm{f}_1\}], [\{\mathrm{f}_0, \mathrm{f}_1, \bar{\mathrm{x}}\}]\}$ the relations follow immediately from Theorem 11 in Lagerkvist [16]. Hence, the remaining cases are when $\mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$, $\mathrm{Pol}(\Gamma) = [\bar{\mathrm{x}}]$, and $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0, \mathrm{f}_1\}]$. First assume that $\mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$. From Lagerkvist [16] we know that $R^{\neq\neq\neq 01}_{1/3} \in \langle\Gamma\rangle_{\nexists}$ where $R^{\neq\neq\neq 01}_{1/3} = \{(0,0,1,1,1,0,0,1),(0,1,0,1,0,1,0,1),(1,0,0,0,1,1,0,1)\}$, and using this relation we can qfpp-define $\tau^{01}_{\neq}$ as

$$\tau^{01}_{\neq}(x_1,x_2,x_3,x_4) \equiv R^{\neq\neq\neq 01}_{1/3}(x_1,x_2,x_3,x_2,x_1,x_4,x_3,x_4)$$

and τ^{01} as $\tau^{01}(x_1,x_2) \equiv \tau^{01}_{\neq}(x_1,x_2,x_1,x_2)$. Now assume that $\mathrm{Pol}(\Gamma) = [\{\bar{\mathrm{x}}\}]$. In this case it is known that the relation $R^{\neq\neq\neq\neq}_{2/4} = R^{\neq\neq\neq 01}_{1/3} \cup \{\bar{t} \mid t \in R^{\neq\neq\neq 01}_{1/3}\}$ is qfpp-definable by Γ [13, 16]. Using this relation one can verify that $\tau_{\neq}(x_1,x_2) \equiv R^{\neq\neq\neq\neq}_{2/4}(x_1,x_1,x_2,x_2,x_2,x_1,x_1,x_2)$. Last, for $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0, \mathrm{f}_1\}]$, the relation $R = \{(0,0,0,0),(0,0,1,1),(0,1,0,1),(1,1,1,1)\} \in \langle\Gamma\rangle_{\nexists}$ [16], and this relation can qfpp-define $\tau_{\to}$ by $\tau_{\to}(x_1,x_2) \equiv R(x_1,x_1,x_2,x_2)$. ◀

The usefulness of this lemma is that we now have a better understanding of the expressiveness of the languages under consideration. For example, if $\mathrm{Pol}(\Gamma) = [\{\bar{\mathrm{x}}\}]$ then we know that Γ is expressive enough to qfpp-define the binary inequality relation $\tau_{\neq}$. Before we begin to prove Theorem 8 we present a lemma that will simplify some of the forthcoming reductions. If R is an n-ary relation then the ith argument is *redundant* if there exists $j \neq i$ such that $t[i] = t[j]$ for every $t \in R$, and R is said to be *irredundant* if it does not have any redundant arguments. It is not difficult to see that there for any R exists an irredundant relation R^{irr} with the property that $\langle\{R\}\rangle_{\nexists} = \langle\{R^{\mathrm{irr}}\}\rangle_{\nexists}$, and we obtain the following lemma.

▶ **Lemma 10.** *Let* Γ *be a constraint language and* R *an* n*-ary relation. Then* $R \in \langle\Gamma\rangle_{\nexists}$ *if and only if* $R^{\mathrm{irr}} \in \langle\Gamma\rangle_{\nexists}$.

In some of the forthcoming reductions we will need the ability to output an arbitrary yes- or no-instance of Inv-SAT(Γ). Clearly, a yes-instance can easily be produced by simply outputting $R \in \Gamma$, but to find $R \notin \langle\Gamma\rangle_{\nexists}$ requires a bit more work. We will provide a proof sketch for how such a relation can be constructed. Begin by enumerating all partial polymorphisms of Γ up to arity $k+1$, where k is the maximum arity of any relation in Γ. It is well-known that any finite Boolean constraint language containing only relations of arity k contains a partial polymorphism which is not a partial projection [18]. Hence, let f denote such a partial polymorphism of arity $n \leq k+1$, and let $\mathrm{domain}(f) = \{t_1,\ldots,t_m\}$. Now consider the relation R obtained by for each $1 \leq i \leq n$ adding the tuple $(t_1[i],\ldots,t_m[i])$. By construction, f does not preserve R since it is not a subfunction of a projection, which by Theorem 5 implies that $R \notin \langle\Gamma\rangle_{\nexists}$. We are now ready to prove our first result, and begin with the case when $\mathrm{Pol}(\Gamma)$ consists only of projections (which due to Theorem 4 implies that Γ can pp-define every Boolean relation).

▶ **Lemma 11.** *Let* Γ *be a finite constraint language such that* $\mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$. *Then* Inv-SAT($\Gamma$) *is co-NP-complete.*

Proof. First consider the constraint language $\Delta = \{\tau \times \{(0,1)\}\} \mid \tau \in \Gamma\} \cup \{\{(0)\}, \{(1)\}$, i.e., each relation in Γ is adjoined with two constant arguments, and in addition Δ contains both $\{(0)\}$ and $\{(1)\}$. Since Δ is ultraidempotent and $\mathrm{Pol}(\Delta) = \mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$ it follows from Theorem 6 that Inv-SAT(Δ) is co-NP-complete. Hence, let R be an n-ary relation, i.e., an instance of Inv-SAT(Δ). We now observe that if there exists $1 \leq i \leq n$ such that $\mathrm{Pr}_i(R) = \{(0)\}$ but no $1 \leq j \leq n$ such that $\mathrm{Pr}_j(R) = \{(1)\}$, then $R \in \langle\Delta\rangle_{\not\exists}$ if and only if $R \in \langle\{(0)\}\rangle_{\not\exists}$. Similarly, if there exists $1 \leq i \leq n$ such that $\mathrm{Pr}_i(R) = \{(1)\}$ but no $1 \leq j \leq n$ such that $\mathrm{Pr}_j(R) = \{(0)\}$, then $R \in \langle\Delta\rangle_{\not\exists}$ if and only if $R \in \langle\{(1)\}\rangle_{\not\exists}$. In both these cases we can compute the answer in polynomial time and output an arbitrary yes- or no-instance to Inv-SAT(Γ).

Hence, assume that there exist both i and j such that $\mathrm{Pr}_i(R) = \{(0)\}$ and $\mathrm{Pr}_j(R) = \{(1)\}$, and for simplicity assume that R does not contain any redundant arguments, which is possible by Lemma 10. We then claim that $R \in \langle\Delta\rangle_{\not\exists}$ if and only if $R \in \langle\Gamma\rangle_{\not\exists}$. Hence, first assume that $R \in \langle\Delta\rangle_{\not\exists}$, and let $R(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \equiv \varphi(x_1, \ldots, x_n) \wedge \{(0)\}(x_i) \wedge \{(1)\}(x_j)$ be a qfpp-definition witnessing this, where we without loss of generality assume that every constraint in $\varphi(x_1, \ldots, x_n)$ is of the form $\tau_k \times \{(0,1)\}(\mathbf{x}_k, x_i, x_j)$ for $\tau \times \{(0,1)\} \in \Delta$, where $\mathbf{x}_k$ is a tuple of variables of length $\mathrm{ar}(\tau_k)$ not containing x_i or x_j. Then we may obtain a qfpp-definition of R over Γ by first replacing $\{(0)\}(x_i) \wedge \{(1)\}(x_j)$ by the single constraint $\{(0,1)\}(x_i, x_j)$, and then replacing every constraint $\tau_k \times \{(0,1)\}(\mathbf{x}_k, x_i, x_j)$ in $\varphi(x_1, \ldots, x_n)$ by $\tau_k(\mathbf{x}_k)$. This is clearly a valid qfpp-definition of R over Γ since $\{(0,1)\} \in \langle\Gamma\rangle_{\not\exists}$ by Lemma 9. For the other direction, assume that $R \in \langle\Gamma\rangle_{\not\exists}$ and let $R(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \equiv \varphi(x_1, \ldots, x_n)$ be a qfpp-definition of R over Γ, where every constraint in $\varphi(x_1, \ldots, x_n)$ is of the form $\tau(\mathbf{x})$ for $\tau \in \Gamma$. Then we can construct a qfpp-definition of R over Δ by first introducing the constraints $\{(0)\}(x_i)$ and $\{(1)\}(x_j)$, and then replacing every $\tau_k(\mathbf{x}_k)$ by $\tau_k \times \{(0,1)\}(\mathbf{x}_k, x_i, x_j)$. ◀

▶ **Lemma 12.** *Let* Γ *be a finite constraint language such that* $\mathrm{Pol}(\Gamma) = [\{\bar{\mathrm{x}}\}]$. *Then* Inv-SAT($\Gamma$) *is co-NP-complete.*

Proof. We will give a polynomial-time reduction from Inv-SAT($\Gamma \cup \{(0,1)\}$), which is co-NP-complete by Lemma 11, since $\mathrm{Pol}(\Gamma \cup \{(0,1)\}) = \Pi_{\mathbb{B}}$. Hence, let R be an n-ary relation, i.e, an instance of Inv-SAT($\Gamma \cup \{(0,1)\}$). If there exist neither i nor j such that $\mathrm{Pr}_i(R) = \{(0)\}$ and $\mathrm{Pr}_j(R) = \{(1)\}$ then $R \in \langle\Gamma\rangle_{\not\exists}$ if and only if $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$, and the output of the reduction is simply R. Furthermore, if there exists $1 \leq i \leq n$ such that $\mathrm{Pr}_i(R) = \{(0)\}$ but no $1 \leq j \leq n$ such that $\mathrm{Pr}_j(R) = \{(1)\}$, or vice versa, then it cannot be the case that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ or $R \in \langle\Gamma\rangle_{\not\exists}$, which again implies that we may simply output R. This implies that the only remaining case is when there exist both i and j such that $\mathrm{Pr}_i(R) = \{(0)\}$ and $\mathrm{Pr}_j(R) = \{(1)\}$. By Lemma 10 we may without loss of generality assume that R does not contain any other constant arguments.

In this case we claim that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ if and only if $\neg(R) \in \langle\Gamma\rangle_{\not\exists}$, where $\neg(R) = R \cup \{\bar{t} \mid t \in R\}$, i.e., R closed under complement. Assume first that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ and let $R(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m) \wedge \{(0,1)\}(x_i, x_j)$ denote a qfpp-definition over $\Gamma \cup \{(0,1)\}$, where $R_1, \ldots, R_m \in \Gamma$. We will construct a qfpp-definition of $\neg(R)$ over Γ as follows. First, we replace $\{(0,1)\}(x_i, x_j)$ by the constraint $\tau_{\neq}(x_i, x_j)$, which is qfpp-definable over Γ by Lemma 9. Then every other constraint is kept unchanged and we obtain the qfpp-definition $R'(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m) \wedge \tau_{\neq}(x_i, x_j)$ over Γ. We claim that $R' = \neg(R)$. It is easy to see that $\neg(R) \subseteq R'$. Hence, let $t \in R'$,

assume that $t \notin \neg(R)$, and observe that this also implies that $\bar{t} \notin \neg(R)$, since $\neg(R)$ is closed under complement. Due to the construction of R' this is clearly not possible. For the other direction, assume that $\neg(R) \in \langle\Gamma\rangle_{\not\exists}$ and let $\neg(R)(x_1, \ldots, x_n) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$ be a qfpp-definition over Γ where $R_1, \ldots, R_m \in \Gamma$. We can then qfpp-define R using Γ and $\{(0,1)\}$ as $R(x_1, \ldots, x_n) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m) \wedge \{(0,1)\}(x_i, x_j)$, since this only keeps $t \in \neg(R)$ satisfying $t[i] = 0$ and $t[j] = 1$. ◀

For the case when Γ is preserved by a constant operation we will first need to show co-NP-completeness of the following auxiliary problem. Say that an n-ary Boolean relation R is *complementary saturated* if there for every $1 \leq i \leq n$ exists $1 \leq j \leq n$ such that $t[i] \neq t[j]$ for every $t \in R$. In other words for each argument of the relation there exists an argument which is its complement. For a finite constraint language Γ by Inv-SAT$^{\neq}(\Gamma)$ now we denote the structurally restricted Inv-SAT(Γ) problem defined as follows.

Instance: A complementary saturated Boolean relation R. Question: $R \in \langle\Gamma\rangle_{\not\exists}$?

We will now prove that Inv-SAT$^{\neq}(\Gamma)$ remains co-NP-complete when $\mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$.

▶ **Lemma 13.** *Let Γ be a finite constraint language such that $\mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}$. Then* Inv-SAT$^{\neq}(\Gamma)$ *is co-NP-complete.*

Proof. We will first construct the language $\Gamma^{\neq}$ for every $R \in \Gamma$ by letting $R^{\neq} \in \Gamma^{\neq}$ where $R^{\neq}$ is obtained by adding the minimum number of arguments to R such that $R^{\neq}$ is complementary saturated. Without loss of generality we assume that the arguments to each relation $R^{\neq} \in \Gamma^{\neq}$ is ordered in such a way that $\mathrm{Pr}_{1,\ldots,\mathrm{ar}(R)}(R^{\neq}) = R$, and that the remaining arguments are the complement of the arguments of R. Observe that $\mathrm{Pol}(\Gamma^{\neq}) = \Pi_{\mathbb{B}}$, which by Lemma 11 implies that Inv-SAT$(\Gamma^{\neq})$ is co-NP-complete.

Hence, let R be an n-ary relation. Assume that R is not complementary saturated, i.e., not a valid instance of Inv-SAT$^{\neq}(\Gamma)$. Then either $R \in \langle\{(0,0),(1,1)\}\rangle_{\not\exists} \subseteq \langle\Gamma^{\neq}\rangle_{\not\exists}$, in which case we output an arbitrary yes-instance, or $R \notin \langle\Gamma^{\neq}\rangle_{\not\exists}$, in which case we output an arbitrary no-instance. Otherwise R is already a valid instance of Inv-SAT$^{\neq}(\Gamma)$, and in this case we claim that $R \in \langle\Gamma^{\neq}\rangle_{\not\exists}$ if and only if $R \in \langle\Gamma\rangle_{\not\exists}$. First assume that $R \in \langle\Gamma^{\neq}\rangle_{\not\exists}$. Via Lemma 9 we know that $\tau^{01}_{\neq} \in \langle\Gamma\rangle_{\not\exists}$, and from this property it follows that $\Gamma^{\neq} \subseteq \langle\Gamma\rangle_{\not\exists}$, implying that $R \in \langle\Gamma\rangle_{\not\exists}$. For the other direction, assume that $R \in \langle\Gamma\rangle_{\not\exists}$. Let $R(x_1, \ldots, x_n) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$ be a qfpp-definition over Γ, and for each tuple of variables $\mathbf{x}_i$ let $\mathbf{y}_i$ denote the corresponding tuple of complementary variables. It then follows that $R(x_1, \ldots, x_n) \equiv R^{\neq}_1(\mathbf{x}_1, \mathbf{y}_1) \wedge \ldots \wedge R^{\neq}_m(\mathbf{x}_m, \mathbf{y}_m)$ is a valid qfpp-definition of R over $\Gamma^{\neq}$. ◀

▶ **Lemma 14.** *Let Γ be a finite constraint language such that $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0\}]$ or $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_1\}]$. Then* Inv-SAT$(\Gamma)$ *is co-NP-complete.*

Proof. We present the proof for the case when $\mathrm{Pol}(\Gamma = [\{\mathrm{f}_1\}]$ since the other case is entirely analogous. In order to prove this we will give a polynomial-time reduction from Inv-SAT$^{\neq}(\Gamma \cup \{(0,1)\})$ to Inv-SAT(Γ). The problem Inv-SAT$^{\neq}(\Gamma \cup \{(0,1)\})$ is co-NP-complete by Lemma 13 since $\mathrm{Pol}(\Gamma \cup \{(0,1)\}) = \Pi_{\mathbb{B}}$.

Let R be an instance Inv-SAT$^{\neq}(\Gamma \cup \{(0,1)\})$ of arity n. If there does not exist $i, j \in \{1, \ldots, n\}$ such that $\mathrm{Pr}_i(R) = \{(0)\}$ and $\mathrm{Pr}_j(R) = \{(1)\}$ then it is already the case that $R \in \langle\Gamma\rangle_{\not\exists}$ if and only if $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$; therefore we assume that such i and j exist. First

construct the relation $R' = R \cup \{(0,\dots,0)\}$, i.e., the relation R adjoined with the constant 0 tuple. We will now prove that $R' \in \langle\Gamma\rangle_{\not\exists}$ if and only if $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$. Hence, first assume that $R' \in \langle\Gamma\rangle_{\not\exists}$ and let $R'(x_1,\dots,x_n) \equiv R_1(\mathbf{x}_1) \wedge \dots \wedge R_m(\mathbf{x}_m)$ be a qfpp-definition over Γ. Then consider the qfpp-definition $R(x_1,\dots,x_n) \equiv R_1(\mathbf{x}_1) \wedge \dots \wedge R_m(\mathbf{x}_m) \wedge \{(0,1)\}(x_i,x_j)$. The intuition behind this qfpp-definition is that the additional constraint $\{(0,1)\}(x_i,x_j)$ will ensure that the constant 0 tuple included in R' but not in R. For the other direction assume that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ and let $R(x_1,\dots,x_n) \equiv R_1(\mathbf{x}_1) \wedge \dots \wedge R_m(\mathbf{x}_m) \wedge \{(0,1)\}(x_i,x_j)$ denote a qfpp-definition, where we without loss of generality assume that $R_1,\dots,R_m \in \Gamma$. Now recall the relation $\tau_{\neq}^{01} \cup \{(0,0,0,0)\} = \{(0,1,0,1),(1,0,0,1),(0,0,0,0)\}$ from Lemma 9, and observe that this relation is nothing else than the binary inequality relation with two constant arguments, adjoined with the constant 0 tuple. We will use this relation as a gadget in order to enforce that the correct inequalities hold between the complementary variables. Hence, assume that the arity of R is $2k+2$, that the variables occuring in positions $k+1,\dots,2k$ are the complement of the k first, and that the last two arguments are constant 0 and constant 1, respectively. We can thus qfpp-define R' as

$$R'(x_1,\dots,x_k,x_{k+1},\dots,x_{2k},x_{2k+1},x_{2k+2}) \equiv R_1(\mathbf{x}_1) \wedge \dots \wedge R_m(\mathbf{x}_m) \wedge \bigwedge_{i=1}^{k} \tau_{\neq}^{01} \cup \{(0,0,0,0)\}(x_i,x_{i+k},x_{2k+1},x_{2k+2}).$$

To see that this definition is indeed correct, note that if x_i and x_{i+k} are both assigned the value 0, then this also forces the variable x_{2k+2} the value 0. But this implies that every other variable must be assigned 0 as well, yielding the constant 0 tuple which is included in R'. ◀

▶ **Lemma 15.** *Let Γ be a finite constraint language such that $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0,\mathrm{f}_1\}]$. Then* Inv-SAT($\Gamma$) *is co-NP-complete.*

Proof. In order to prove the result we will give a polynomial-time reduction from Inv-SAT($\Gamma \cup \{(0,1)\}$), which is co-NP-complete since $\mathrm{Pol}(\Gamma \cup \{(0,1)\}) = \Pi_{\mathbb{B}}$. Hence, let R be an n-ary relation. If there does not exist $i,j \in \{1,\dots,n\}$ such that $\mathrm{Pr}_i(R) = \{(0)\}$ and $\mathrm{Pr}_j(R) = \{(1)\}$ then it is already the case that $R \in \langle\Gamma\rangle_{\not\exists}$ if and only if $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$. Therefore, assume that such i and j exist, and construct the relation $R' = R \cup \{(0,\dots,0),(1,\dots,1)\}$. We claim that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ if and only if $R' \in \langle\Gamma\rangle_{\not\exists}$. For the first direction, assume that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ and let $R(x_1,\dots,x_n) \equiv R_1(\mathbf{x}_1) \wedge \dots \wedge R_m(\mathbf{x}_m) \wedge \{(0,1)\}(x_i,x_j)$ denote a qfpp-definition such that $R_1,\dots,R_m \in \Gamma$. Recall that $\tau_{\to} = \{(0,0),(0,1),(1,1)\}$ from Lemma 9 is qfpp-definable by Γ. Now construct the qfpp-definition

$$R'(x_1,\dots,x_n) \equiv R_1(\mathbf{x}_1) \wedge \dots \wedge R_m(\mathbf{x}_m) \wedge \bigwedge_{k=1}^{n} (\tau_{\to}(x_i,x_k) \wedge \tau_{\to}(x_k,x_j).$$

To see that this definition is correct, observe that the additional constraints of the form $(\tau_{\to}(x_i,x_k) \wedge \tau_{\to}(x_k,x_j)$ ensure that either x_i and x_j are assigned 0 and 1, respectively, or every variable is assigned 0 or 1, resulting in the two constant tuples $(0,\dots,0)$ and $(1,\dots,1)$. The other direction ($R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\exists}$ if $R' \in \langle\Gamma\rangle_{\not\exists}$) can be proven using similar arguments as in the proof of Lemma 14. ◀

▶ **Lemma 16.** *Let Γ be a finite constraint language such that $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0,\mathrm{f}_1,\bar{\mathrm{x}}\}]$. Then* Inv-SAT($\Gamma$) *is co-NP-complete.*

Proof. As $\mathrm{Pol}(\Gamma) = [\{\mathrm{f}_0, \mathrm{f}_1, \bar{\mathrm{x}}\}]$ it follows that $\mathrm{Pol}(\Gamma \cup \{(0,1)\}) = \Pi_{\mathbb{B}}$. We will give a polynomial-time reduction from Inv-SAT$^{\neq}(\Gamma \cup \{(0,1)\})$ to Inv-SAT(Γ) (Inv-SAT$^{\neq}(\Gamma \cup \{(0,1)\})$ is co-NP-complete since $\mathrm{Pol}(\Gamma \cup \{(0,1)\}) = \Pi_{\mathbb{B}}$). Let R be an instance of Inv-SAT$^{\neq}(\Gamma \cup \{(0,1)\})$. First we check whether there exists i and j such that $\mathrm{Pr}_i(R) = \{(0)\}$ and $\mathrm{Pr}_j(R) = \{(1)\}$. If this is not the case then $R \in \langle\Gamma\rangle_{\not\equiv}$ if and only if $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\equiv}$, and we are done. Therefore assume that such i and j exists. For simplicity we will also assume that R is irredundant, $n = 2k+2$, $i = 2k+1$, $j = 2k+2$, and that the arguments in positions $k+1, \ldots, 2k$ are the complement of the k first. Construct the relation $R' = R \cup \{\bar{t} | t \in R\} \cup \{(0,\ldots,0),(1,\ldots,1)\}$. We will prove that $R' \in \langle\Gamma\rangle_{\not\equiv}$ if and only if $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\equiv}$. Therefore, first assume that $R \in \langle\Gamma \cup \{(0,1)\}\rangle_{\not\equiv}$ and let $R(x_1,\ldots,x_{2k+2}) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m) \wedge \{(0,1)\}(x_{2k-1}, x_{2k})$ be a qfpp-definition over Γ where $R_1, \ldots, R_m \in \Gamma$. Now consider the qfpp-definition

$$R''(x_1,\ldots x_{2k+2}) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m) \wedge \bigwedge_{l \in \{1,\ldots,n\}} \tau_{f_0,f_1,\bar{x}}(x_l, x_{l+k}, x_{2k+1}, x_{2k+2}),$$

where $\tau_{f_0,f_1,\bar{x}} = \{(0,0,0,0),(1,1,1,1),(0,1,0,1),(1,0,0,1),(1,0,1,0),(0,1,1,0)\} \in \langle\Gamma\rangle_{\not\equiv}$ is the relation from Lemma 9. We claim that $R' = R''$, i.e., then the above qfpp-definition defines R'. It is clearly the case that $R \subseteq R''$, and this also implies that $R' \subseteq R''$ since R'' is closed under f_0, f_1, and $\bar{x}$. For the other direction, assume there exists $t \in R'' \setminus R'$. It must then be the case that t is not constant 0 or constant 1, and furthermore also that $\bar{t} \notin R'$. Assume first that there exists $1 \leq l \leq k$ such that $t[l] = t[l+k]$. Then, due to the constraints $\bigwedge_{l \in \{1,\ldots,n\}} \tau_{f_0,f_1,\bar{x}}(x_l, x_{l+k}, x_{2k+1}, x_{2k+2})$, it is easy to verify that this will force $t[2k+1] = t[2k+2] = t[l]$, which in turn implies that $t[l] = t[l']$ for every $1 \leq l' \leq 2k+2$, and also that $t \in R$. This contradicts the assumption, and we conclude (1) that $t[l] \neq t[l+k]$ for every $1 \leq l \leq k$ and (2) that $t[2k+1] = 0$ and $t[2k+2] = 1$ or $t[2k+1] = 1$ and $t[2k+2] = 0$. In the first case it directly follows that $t \in R \subseteq R'$, and in the second case that $\bar{t} \in R$, and hence that $t \in R'$.

To prove the reverse direction we assume that $R'(x_1,\ldots x_{2k+2}) \equiv R_1(\mathbf{x}_1) \wedge \ldots \wedge R_m(\mathbf{x}_m)$ where $R_1, \ldots, R_m \in \Gamma$. We can then qfpp-define R by $R(x_1,\ldots x_{2k+2}) \equiv R'(x_1,\ldots x_{2k+2}) \wedge \{(0,1)\}(x_{2k+1}, x_{2k+2})$. This concludes the reduction. ◀

By combining Lemma 11– 12 and Lemma 14– 16 we have thus finally proven Theorem 8.

▶ **Example 17.** We can now answer the question regarding the complexity of Inv-SAT$(\{R_{1/3}\})$ from Example 3. It is not hard to verify that $R_{1/3}$ is only preserved by the projections, from which it follows that $\mathrm{Pol}(\mathrm{R}_{1/3}) = \Pi_{\mathbb{B}}$. An application of Theorem 8 then reveals that Inv-SAT$(\{R_{1/3}\})$ is indeed co-NP-complete.

4 The Inv-SAT(Γ) Problem over Infinite Constraint Languages

Since we have proven that Inv-SAT(Γ) is always either tractable or co-NP-complete for finite Γ, it is tempting to investigate the case when Γ is infinite. First, it is important to note that Schaefer's dichotomy theorem for SAT(Γ) is also valid for infinite constraint languages, and in fact that many natural satisfiability problems such as CNF-SAT, Horn-SAT, and linear equations modulo 2, can only be represented as SAT(Γ) problems over infinite Γ. It thus makes sense to ask whether it is possible to extend Theorem 8 to infinite constraint languages. First, note that if SAT(Γ) is NP-complete then SAT(Δ) is NP-complete whenever $\Delta \subseteq \Gamma$. This straightforward property does *not* hold for Inv-SAT(Γ), since, for example, Inv-SAT$(\{R_{1/3}\})$ is co-NP-complete but Inv-SAT(BR) is trivially solvable in polynomial

time by always answering "yes". We will now describe a more general class of tractable Inv-SAT(Γ) problems based on properties of the partial polymorphisms of Γ.

▶ **Theorem 18.** *Let* Γ *be a constraint language such that* pPol(Γ) *admits a finite base* F*. Then* Inv-SAT(Γ) *is solvable in polynomial time.*

Proof. Let R be an instance of Inv-SAT(Γ) of arity n. Due to the Galois connection in Theorem 5 the question $R \in \langle\Gamma\rangle_{\nexists}$ is equivalent to checking whether $F \subseteq \mathrm{pPol}(\{\mathrm{R}\})$, or, put otherwise, whether R is preserved by every partial operation in F. Now consider the following algorithm.

1. Let k be the maximum arity among the partial operations in F.
2. For each $1 \leq i \leq k$ enumerate all sequences $t_1, \ldots, t_i \in R$.
3. For each $f \in F$ of arity i compute $f(t_1, \ldots, t_i) = t$. If $t \notin R$ then answer "no".
4. Answer "yes".

As remarked, this algorithm is sound and complete since $R \in \langle\Gamma\rangle_{\nexists}$ if and only if every $f \in F$ preserves R, and an i-ary partial operation f preserves R if and only if there does not exist $t_1, \ldots, t_i \in R$ such that $f(t_1, \ldots, t_i) \notin R$. Regarding the time complexity, we in the ith iteration enumerate all sequences of tuples from R of length i, which takes $O(|R|^i)$ time, and for each $f \in F$ check whether f applied to this sequence results in a tuple included in R, which takes $O(i \cdot n \cdot |R|)$ time. Put together this gives a running time of $O(k \cdot |F| \cdot |R|^k \cdot k \cdot n \cdot |R|) = O(k^2 \cdot |F| \cdot |R|^{k+1} \cdot n)$ which is bounded by a polynomial since k is a fixed constant. ◀

It is worth remarking that Γ is *always* infinite when pPol(Γ) is finitely generated and $\mathrm{Pol}(\Gamma) \supseteq [\{\mathrm{f}_0, \mathrm{f}_1, \overline{\mathrm{x}}]$ [17] — hence there is no possible overlap between Theorem 8 and Theorem 18. This result may be seen as surprising since computational problems parameterized by Boolean constraint languages tend to be rather well-behaved, and to the best of our knowledge only a variant of the propositional abduction problem exhibits a similar difference in complexity between finite and infinite constraint languages [12]. At this stage it is fair to ask if Inv-SAT(Γ) is always tractable when Γ is infinite. This is however not the case. First take any finite constraint language Γ such that Inv-SAT(Γ) is co-NP-complete by Theorem 8. Then consider the infinite constraint language $\langle\Gamma\rangle_{\nexists}$ obtained by closing Γ under qfpp-definitions. Clearly, Inv-SAT(Γ) and Inv-SAT($\langle\Gamma\rangle_{\nexists}$) are the same computational problem, and in particular Inv-SAT($\langle\Gamma\rangle_{\nexists}$) is co-NP-complete even though $\langle\Gamma\rangle_{\nexists}$ is infinite. Based on these observations and Theorem 18, it is natural to conjecture that the question of whether Inv-SAT(Γ) is co-NP-complete or tractable does not depend on whether Γ is finite or infinite, but rather whether pPol(Γ) is sufficiently simple. We thus make the following conjecture.

▶ **Conjecture 19.** *Let* Γ *be a Boolean constraint language such that* $\mathrm{Pol}(\Gamma) \supseteq [\{\mathrm{f}_0, \mathrm{f}_1, \overline{\mathrm{x}}\}]$*. Then* Inv-SAT(Γ) *is tractable if* pPol(Γ) *is finitely generated and is co-NP-hard otherwise.*

5 Concluding Remarks

We have studied the complexity of Inv-SAT(Γ) and obtained a complete dichotomy theorem for finite Γ. To prove this we first limited the number of cases we needed to consider with polymorphisms, and for each such case then used expressibility results based on partial polymorphisms, in order to proceed with the required reductions. We also demonstrated that Inv-SAT(Γ) is also a relevant problem for infinite constraint languages, even though the situation differs drastically from the finite case. These results raise a few different directions for future research.

A Dichotomy Theorem for Infinite Constraint Languages

A good starting point for proving Conjecture 19 is to find examples of infinite Γ such that (1) there does not exist any finite $\Delta \subset \Gamma$ such that $\langle\Gamma\rangle_{\not\exists} = \langle\Delta\rangle_{\not\exists}$ and (2) pPol(Γ) is infinitely generated. One candidate for such a language is $\Gamma_{\text{XSAT}} = \{R_{1/k} \mid k \geq 3\}$, $R_{1/k} = \{(b_1, \ldots, b_k) \in \{0,1\}^k \mid b_1 + \ldots + b_k = 1\}$, where both these properties can be proven to hold. Is Inv-SAT(Γ_{XSAT}) tractable or co-NP-complete?

The Inverse Constraint Satisfaction Problem

The *constraint satisfaction problem* over a constraint language Γ (CSP(Γ)) is a multi-valued generalization of SAT where Γ may contain non-Boolean relations. One may then define Inv-CSP(Γ) analogously to Inv-SAT and ask if a dichotomy theorem can be obtained for finite Γ. This is likely a good deal harder than the Boolean case and a starting point would be to compare the complexity of Inv-CSP(Γ) to the complexity of enumerating solutions of Inv-CSP(Γ) with polynomial delay [22]. In particular it would be interesting to find examples of Γ such that Inv-CSP(Γ) is tractable even though the enumeration problem is not tractable.

Another tempting problem is to study Inv-CSP(Γ) over infinite domains. In this case some extra care is needed since the instance R cannot always be represented explicitly as a list of tuples. However, there exist well-studied, so called *ω-categorical*, constraint languages where the Inv-CSP problem could be interesting, since there exist better methods to represent relations than listing its tuples. However, even the problem of checking if $R \in \langle\Gamma\rangle$ for Γ over infinite domains is in general undecidable [2], so there is little hope in obtaining a complete dichotomy.

Acknowledgements. We thank the anonymous reviewers for several helpful comments.

References

1. L. Barto. Constraint satisfaction problem and universal algebra. *ACM SIGLOG News*, 1(2):14–24, 2014.
2. M. Bodirsky, M. Pinsker, and T. Tsankov. Decidability of definability. *Journal of Symbolic Logic*, 78(4):1036–1054, 2013.
3. V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I. *Cybernetics*, 5:243–252, 1969.
4. V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. II. *Cybernetics*, 5:531–539, 1969.
5. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *ACM SIGACT-Newsletter*, 34(4):38–52, 2003.
6. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part II: Constraint satisfaction problems. *ACM SIGACT-Newsletter*, 35(1):22–35, 2004.
7. H. Chen. Inverse NP problems. *Computational Complexity*, 17(1):94–118, 2008.
8. N. Creignou and H. Vollmer. Boolean constraint satisfaction problems: When does Post's lattice help? In N. Creignou, P. G. Kolaitis, and H. Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 3–37. Springer Berlin Heidelberg, 2008.
9. Nadia Creignou, Phokion G. Kolaitis, and Bruno Zanuttini. Structure identification of boolean relations and plain bases for co-clones. *J. Comput. Syst. Sci.*, 74(7):1103–1115, 2008. doi:10.1016/j.jcss.2008.02.005.

10 Creignou, N. and Hebrard, J.-J. On generating all solutions of generalized satisfiability problems. *RAIRO-Theor. Inf. Appl.*, 31(6):499–511, 1997.

11 D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27(1):95–100, 1968.

12 P. Jonsson, V. Lagerkvist, and G. Nordh. Constructing np-intermediate problems by blowing holes with parameters of various properties. *Theoretical Computer Science*, 581:67–82, 2015.

13 P. Jonsson, V. Lagerkvist, G. Nordh, and B. Zanuttini. Strong partial clones and the time complexity of SAT problems. *Journal of Computer and System Sciences*, 84:52–78, 2017.

14 D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28:152–163, 1998.

15 R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.

16 V. Lagerkvist. Weak bases of Boolean co-clones. *Information Processing Letters*, 114(9):462–468, 2014.

17 V. Lagerkvist and M. Wahlström. The power of primitive positive definitions with polynomially many variables. *Journal of Logic and Computation*, 27(5):1465–1488, 2017.

18 V. Lagerkvist, M. Wahlström, and B. Zanuttini. Bounded bases of strong partial clones. In *Proceedings of the 45th International Symposium on Multiple-Valued Logic (ISMVL-2015)*, pages 189–194, 2015.

19 E. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

20 B.A. Romov. The algebras of partial functions and their invariants. *Cybernetics*, 17(2):157–167, 1981.

21 T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory Of Computing (STOC-78)*, pages 216–226. ACM Press, 1978.

22 H. Schnoor and I. Schnoor. Enumerating all solutions for constraint satisfaction problems. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS-2007)*, volume 4393, pages 694–705. Springer, 2007.

23 H. Schnoor and I. Schnoor. Partial polymorphisms and constraint satisfaction problems. In N. Creignou, P. G. Kolaitis, and H. Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 229–254. Springer Berlin Heidelberg, 2008.

24 R. Willard. Testing expressibility is hard. In *Proceedings of the 16th International Conference Principles and Practice of Constraint Programming (CP-2010)*, volume 6308 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2010.

Balanced Judicious Bipartition is Fixed-Paramater Tractable*

Daniel Lokshtanov[1], Saket Saurabh[2], Roohani Sharma[3], and Meirav Zehavi[4]

1 **Department of Informatics, University of Bergen, Norway**
daniello@uib.no
2 **Department of Informatics, University of Bergen, Norway and Institute of Mathematical Sciences, HBNI, Chennai, India and UMI ReLax**
saket@imsc.res.in
3 **Institute of Mathematical Sciences, HBNI, Chennai, India and UMI ReLax**
roohani@imsc.res.in
4 **Department of Informaics, University of Bergen, Norway**
meirav.zehavi@uib.no

Abstract

The family of judicious partitioning problems, introduced by Bollobás and Scott to the field of extremal combinatorics, has been extensively studied from a structural point of view for over two decades. This rich realm of problems aims to counterbalance the objectives of classical partitioning problems such as Min Cut, Min Bisection and Max Cut. While these classical problems focus solely on the minimization/maximization of the number of edges crossing the cut, judicious (bi)partitioning problems ask the natural question of the minimization/maximization of the number of edges lying in the (two) sides of the cut. In particular, Judicious Bipartition (JB) seeks a bipartition that is "judicious" in the sense that neither side is burdened by too many edges, and Balanced JB also requires that the sizes of the sides themselves are "balanced" in the sense that neither of them is too large. Both of these problems were defined in the work by Bollobás and Scott, and have received notable scientific attention since then. In this paper, we shed light on the study of judicious partitioning problems from the viewpoint of algorithm design. Specifically, we prove that BJB is FPT (which also proves that JB is FPT).

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity, I.1.2 Algorithms, G.2.2 Graph Theory

Keywords and phrases Judicious Partition, Tree Decomposition, Parameterized Complexity, Odd Cycle Transversal, Minimum Bisection

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.40

1 Introduction

More than twenty years ago, Bollobás and Scott [3] defined the notion of *judicious partitioning problems*. Since then, the family of judicious partitioning problems has been extensively studied in the field of Extremal Combinatorics, as can be evidenced by the abundance of structural results described in surveys such as [7, 35]. This rich realm of problems aims to counterbalance the objectives of classical partitioning problems such as Min Cut, Min Bisection, Max Cut and Max Bisection. While these classical problems focus

* A full version of the paper is available at [27], https://arxiv.org/abs/1710.05491.

solely on the minimization/maximization of the number of edges crossing the cut, judicious (bi)partitioning problems ask the natural questions of the minimization/maximization of the number of edges lying in the (two) sides of the cut. Another significant feature of judicious partitioning problems that also distinguishes them from other classical partitioning problems is that they inherently and naturally encompass several objectives, aiming to minimize (or maximize) the number of edges in several sets *simultaneously*.

In this paper, we shed light on properties of judicious partitioning problems from the viewpoint of the design of algorithms. Up until now, the study of such problems has essentially been overlooked at the algorithmic front, where one of the underlying reasons for this discrepancy might be that standard machinery does not seem to handle them effectively. Specifically, we focus on the Judicious Bipartition problem, where we seek a bipartition that is "judicious" in the sense that neither side is burdened by too many edges, and on the Balanced Judicious Bipartition problem, where we also require that the sizes of the sides themselves are "balanced" in the sense that neither of them is too large. Both of these problems were defined in the work by Bollobás and Scott, and have received notable scientific attention since then. Formally, Balanced Judicious Partition is defined as follows.

Balanced Judicious Bipartition (BJB)	**Parameter:** $k_1 + k_2$
Input: A multi-graph G, and integers μ, k_1 and k_2	
Question: Does there exist a partition (V_1, V_2) of $V(G)$ such that $\lvert V_1\rvert = \mu$ and for all $i \in \{1, 2\}$, it holds that $\lvert E(G[V_i])\rvert \leq k_i$?	

We note that in the literature, the term BJB refers to the case where $\mu = \lceil \frac{|V(G)|}{2} \rceil$, and hence it is more restricted then the definition above. By dropping the requirement that $|V_1| = \mu$, we get the Judicious Bipartition (JB) problem. By using new crucial insights into these problems on top of the most advanced machinery in Parameterized Complexity to handle partitioning problems,[1] we are able to resolve the question of the Parameterized Complexity of BJB (and hence also of JB). In particular, we prove the following theorem.

▶ **Theorem 1.** BJB *can be solved in time* $2^{k^{\mathcal{O}(1)}} \cdot |V(G)|^{\mathcal{O}(1)}$.

Structural Results

Denote $n = |V(G)|$ and $m = |E(G)|$. To survey several structural results about judicious partitioning problems, we first define the notions of *t-cut* and *max (min) t-judicious partitioning*. Given a partition of $V(G)$ into t parts, a t-cut is the number of edges going across the parts, while a max (min) judicious t-partitioning is the maximum (minimum) number of edges in any of the parts. When $t = 2$, we use the standard terms *bipartite-cut* and *judicious bipartitioning*, respectively. Furthermore, by t-judicious partitioning we mean max t-judicious partitioning. As stated earlier, Bollobás and Scott [3] defined the notion of *judicious partitioning problems* in 1993. In that paper, they showed that for any positive integer t and graph G, we can partition $V(G)$ into t sets, $V_1, \ldots, V_t$, so that $|E(G[V_i])| \leq \frac{t}{t+1}m$ for all $i \in \{1, \ldots, t\}$. Bollobás and Scott also studied this problem on graphs of maximum degree Δ, and showed that there exists a partition of $V(G)$ into t sets $V_1, \ldots, V_t$ so that it simultaneously satisfies an upper bound and a lower bound on the number of edges in each part as well as on edges between every pair of parts. Later,

[1] To the best of our knowledge, up until now, this machinery has actually only been proven useful to solve one natural problem which could not have been tackled using earlier tools.

Bollobás and Scott [7] gave several new results, leaving open other new questions around judicious partitioning. In [8] they showed an optimal bound for judicious partitioning on bounded-degree graphs. These problems have also been studied on general hypergraphs [4], uniform hypergraphs [23], 3-uniform hypergraphs [6] and directed graphs [25].

The special cases of judicious partitioning problems called judicious bipartitioning and balanced judicious bipartitioning problems have also been studied intensively. Bollobás and Scott [5] proved an upper bound on judicious bipartitioning and proved that every graph that achieves the essentially best known lower bound on bipartite-cut, given by Edwards in [17] and [18], also achieves this upper bound for judicious bipartitioning. In fact, they showed that this is exact for complete graphs of odd order, which are the only extremal graphs without isolated vertices. Alon et al. [1] gave a non-trivial connection between the size of a bipartite-cut in a graph and judicious partitioning into two sets. In particular, they showed that if a graph has a bipartite-cut of size at least $\frac{m}{2} + \delta$ where $\delta \leq m/30$, then there exists a bipartition (V_1, V_2) of $V(G)$ such that $|E(G[V_i])| \leq \frac{m}{4} - \frac{\delta}{2} + \frac{10\delta^2}{m} + 3\sqrt{m}$ for $i \in \{1, 2\}$. They complemented these results by showing an upper bound on the number of edges in each part when $\delta > m/30$. Bollobás and Scott [9] studied similar relations between t-cuts and t-judicious partitionings for $t \geq 3$. Recently, these results were further refined [38, 28]. Xu et al. [37] and Xu and Yu [39] studied balanced judicious bipartitioning where both parts are of almost equal size (that is, one of the sizes is $\lceil \frac{n}{2} \rceil$). Both of these papers concern the following conjecture of Bollobás and Scott [7]: if G is a graph with minimum degree of at least 2, then $V(G)$ admits a balanced bipartition (V_1, V_2) such that for each $i \in \{1, 2\}$, $|E(G[V_i])| \leq \frac{m}{3}$. For further results on judicious partitioning, we refer to the surveys [7, 35].

Algorithmic Results

While classical partitioning problems such as Min Cut, Min Bisection, Max Cut and Max Bisection have been studied extensively algorithmically, the same is not true about judicious partitioning problems. Apart from Min Cut, all the above mentioned partitioning problems are NP-complete. These NP-complete partitioning problems were investigated by all algorithmic paradigms meant for coping with NP-complete, including approximation algorithms and parameterized complexity. In what follows, we discuss known results related to these problems in the realm of parameterized complexity.

First, note that for every graph G, there always exists a bipartition of the vertex set into two parts (in fact equal parts [21, Corollory 1]) such that at least $m/2$ edges are going across. This immediately implies that Max Cut and Max Bisection are FPT when parameterized by the cut size (the number of edges going across the partition). This led Mahajan and Raman [29] to introduce the notion of above-guarantee parameterization. In particular, they showed that one can decide whether a graph has a bipartite-cut of size $\frac{m}{2} + k$ in time $\mathcal{O}(m + n + k4^k)$. However, Edwards [17] showed that every connected graph G has a bipartite-cut of size $\frac{m}{2} + \frac{n-1}{4}$. Thus, a more interesting question asks whether finding a bipartite-cut of size at least $\frac{m}{2} + \frac{n-1}{4} + k$ is FPT. Crowston et al. [15] showed that indeed this is the case as they design an algorithm with running time $\mathcal{O}(8^k n^4)$. Recently, Etscheid and Mnich [19] discovered a kernel with a linear number of vertices (improving upon a kernel by Crowston et al. [14]), and the aforementioned algorithm was sped-up to run in time $\mathcal{O}(8^k m)$ [19]. Gutin and Yeo studied an above-guarantee version of Max Bisection [21], proving that finding a balanced bipartition such that it has at least $\frac{m}{2} + k$ edges is

FPT (also see [33]).[2] In this context Max Bisection, it is also relevant to mention the $(k, n-k)$-Max Cut, which asks for a bipartite-cut of size at least p where one of the sides is of size exactly k. Parameterized by k, this problems is W[1]-hard [11], but parameterized by p, this problem is solvable in time $\mathcal{O}^*(2^p)$ [34] (this result improved upon algorithms given in [10, 36]).

Until recently, the parameterized complexity of Min Bisection was open. Approaches to tackle this problem materialized when the parameterized complexity of ℓ-Way Cut was resolved. Here, given a graph G and positive integers k and ℓ, the objective is to delete at most k edges from G such that it has at least ℓ components. Kawarabayashi and Thorup [24] showed that this problem is FPT. Later, Chitnis et al. [13] developed a completely new tool based on this, called *randomized contractions*, to deal with plethora of cut problems. Other cut problems that have been shown to be FPT include the generalization of Min Cut to Multiway Cut and Multicut [12, 31, 32]. Eventually, Cygan et al. [16], combining ideas underlying the algorithms developed for Multiway Cut, Multicut, ℓ-Way Cut and randomized contractions together with a new kind of decomposition, showed Min Bisection to be FPT. Finally, let us also mention the min c-judicious partitioning (which is a maximization problem), called c-Load Coloring, where given a graph G and a positive integer k, the goal is to decide whether $V(G)$ can be partitioned into c parts so that each part has at least k edges. Barbero et al. [2] showed that this problem is FPT (also see [20]).

Despite the abundance of work described above, the parameterized complexity of JB and BJB has not yet considered. We fill this gap in our studies by showing that both of these problems are FPT. It is noteworthy to remark that one can show that the generalization of Min Bisection to c-Min Bisection, where the objective is to find a partition into c-parts such that each part are almost equal and there are at most k edges going across different parts, is FPT [16]. However, such a generalization is not possible for either JB or BJB. Indeed, even the existence of an algorithm with running time $n^{f(k)}$, for any arbitrary function f, would imply a polynomial-time algorithm for 3-Coloring, where k is set to 0.

Our Approach

For the sake of readability, our strategy of presentation of our proof consists of the definition of a series of problems, each more "specialized" (in some sense) than the previous one, where each section shows that to eventually solve BJB, it is sufficient to focus on some such problem rather than the previous one. We start by showing that we can focus on the solution of the case of BJB where the input graph is bipartite at the cost of the addition of annotations. For this purpose, we present a (not complicated) Turing reduction that employes a known algorithm for the Odd Cycle Transversal problem (see Section 3). The usefulness of the ability to assume that the input graph is bipartite is a key insight in our approach. In particular, the technical parts of our proof crucially rely on the observation that a connected bipartite graph has only two bipartitions (here, we consider bipartitions as ordered pairs). Keeping this intuition in mind, our next step is to reduce the current annotated problem to one where the input graph is also assumed to be connected (this specific argument relies on a simple application of dynamic programming).

Having at hand an (annotated) problem where the input graph is assumed to be a connected bipartite graph, we proceed to the technical part of our proof, which employs the (heavy) machinery developed by Cygan et al. [16]. While this machinery primarily

[2] We refer to surveys [30, 22] for details regarding above-guarantee parameterizations.

aims to tackle problems where one seeks small cuts in addition to some size constraint, our problem involves a priori seemingly different type of constraints. Nevertheless, we observe that once we handle a connected graph, the removal of any set of k edges (to deal with the size constraint and annotations) would not break the graph to more than $k+1$ connected components, and each of these components would clearly be a bipartite graph. Hence, we can view (in some sense) our problem as a cut problem. In practice, the relation between our problem and a cut problem is quite more intricate, and to realize our idea, we crucially rely on the fact that the connected components are bipartite graphs, which allows us to "guess" a binary vector specifying the biparition of their vertex sets in the final solution. This operation entitles the employment of coloring functions (employing $k+1$ colors) and their translation into bipartitions (which at a certain point in our paper, we would start viewing as colorings employing two colors). Let us remark that the machinery introduced by [16] is the computation of a special type of tree decomposition. Accordingly, our approach would eventually involve the introduction of a specialization of BJB that aims to capture the work to perform when handling a bag of the tree decomposition. The definition of this specific problem is very technical, and hence we defer the description of related intuitive explanations to the appropriate locations in Section 5, where we have already set up the required notations to discuss it.

Note that the proofs of statements marked by ($\star$) are omitted due to space constraints and can be found in the full version of the paper [27].

2 Preliminaries

Let $f : A \to B$ be some function. Given $A' \subseteq A$, the notation $f(A') = b$ indicates that for all $a \in A'$, it holds that $f(a) = b$. An *extension* f' of the function f is a function whose domain A' is a superset of A and whose range is B, such that for all $a \in A$, it holds that $f'(a) = f(a)$. For any $A' \subseteq A$, the *restriction* $f|_{A'}$ of f is a function from A' to B such that for any $a \in A'$, $f|_{A'}(a) = f(a)$. Bold face lower-case letters are used to denote tuples (vectors). For any tuple $\mathbf{v}$, we let $\mathbf{v}[i]$ denote the ith coordinate of $\mathbf{v}$. Given some condition ψ, we define $[\psi] = 1$ if ψ is true and $[\psi] = 0$ otherwise. For any positive integer x, we denote by $[x]$ the set $\{1, 2, \ldots, x\}$ and by $[x]_0$ the set $\{0, 1, \ldots, x\}$.

Given a graph G, we let $V(G)$ and $E(G)$ denote the vertex-set and the edge-set of G, respectively. For a subset $A \subseteq V(G)$, we denote by $\delta(A)$ the set of boundary vertices of A, that is, $\delta(A) = \{v \in A : \text{there exists } u \in V(G) \setminus A \text{ such that } \{u, v\} \in E(G)\}$. We let $G \setminus A$ denote the subgraph of G induced by $V(G) \setminus A$. A *bipartite graph* is a graph G such that there exists a bipartition (X, Y) of $V(G)$ where X and Y are independent sets. In this paper, we treat such bipartitions as *ordered pairs*. That is, if (X, Y) is a bipartition of some bipartite graph G, then (Y, X) is assumed to be a *different* bipartition of the graph G. For *connected* bipartite graphs, we have the following simple yet powerful insight.

▶ **Proposition 1** (Folklore). *Any connected bipartite graph G has exactly 2 bipartitions, (X, Y) and (Y, X).*

The *treewidth* of a graph aims to measure how close the graph is to a tree. Formally, this notion is defined as follows. A *tree decomposition* of a graph G is a pair (T, β) such that T is a rooted tree, $\beta : V(T) \to 2^{V(G)}$, and the following conditions are satisfied.

1. For all $\{u, v\} \in E(G)$, there exists $t \in V(T)$ such that $u, v \in \beta(t)$.
2. For all $v \in V(G)$, the subgraph of T induced by $X_v = \{t : v \in \beta(t)\}$ is a (connected) subtree of T on at least one node.

Given $t, \widehat{t} \in V(G)$, the notation $\widehat{t} \preceq t$ indicates that $\widehat{t}$ is a descendant of t in T. Note that t is a descendant of itself. For any $t \in V(T)$, let t' denote the unique parent of t in T. We also need the standard notations $\sigma(t) = \beta(t) \cap \beta(t')$ and $\gamma(t) = \bigcup_{\widehat{t} \preceq t} \beta(\widehat{t})$.

▶ **Proposition 2** (Folklore). *Let (T, β) be a tree decomposition of a graph G. Given a node $t \in V(T)$, let $t_1, \ldots, t_s$ denote the children of t in T, and for all $i \in [s]$, define $V_{t_i} = \gamma(t_i) \setminus \beta(t)$. Let $V_{t'} = V(G) \setminus (\beta(t) \cup \bigcup_{i=1}^{s} V_{t_i})$. Then, the vertex-set of each connected component of $G \setminus \beta(t)$ is a subset of one of $V_{t_1}, \ldots, V_{t_s}, V_{t'}$.*

Let H be some hypergraph. A s *panning forest* of H is a subset $E' \subseteq E(H)$ of minimum size such that the set containing all endpoints of the hyperedges in E' is equal to $V(H)$. In this paper, we implicitly assume that hypergraphs contain no isolated vertices.

A *separation* of a graph G is a pair (X, Y) such that $X, Y \subseteq V(G)$ and $X \cup Y = V(G)$. The *order of a separation* (X, Y) is equal to $|X \cap Y|$. Let G be a graph, $A \subseteq V(G)$, and $q, k \in \mathbb{N}$. The set A is said to be (q, k)*-unbreakable* in G if for *every* separation (X, Y) of G of order at most k, either $|(X \setminus Y) \cap A| \leq q$ or $|(Y \setminus X) \cap A| \leq q$. We also define a notion of unbreakability in the context of functions. A function $g : U \to [k]_0$ is called (q, k)*-unbreakable* if there exists $i \in [k]_0$ such that $\sum_{j \in [k]_0 \setminus \{i\}} |g^{-1}(j)| \leq q$. Let us now claim that there do no exist "too many" (q, k)-unbreakable functions.

▶ **Lemma 2** ($\star$). *For all $q, k \in \mathbb{N}$, the number of (q, k)-unbreakable functions from a universe U to $[k]_0$ is upper bounded by $\sum_{l=0}^{q} \binom{|U|}{l} \cdot q^k \cdot (k+1)$.*

3 Solving Balanced Judicious Bipartition

In this section, we prove Theorem 1 under the assumption that we are given an algorithm for an annotated, yet restricted, variant of BJB. Throughout this section, an instance of BJB is denoted by $\text{BJB}(G, \mu, k_1, k_2)$, and we define $k = k_1 + k_2$. Given a partition (V_1, V_2) that witnesses that an instance $\text{BJB}(G, \mu, k_1, k_2)$ is a YES-instance, we think of the vertices in V_1 as colored 1 and the vertices in V_2 as colored 2; hence, we call such a partition a *witnessing coloring* of $\text{BJB}(G, \mu, k_1, k_2)$. To prove Theorem 1, we first define the Odd Cycle Transversal problem. Here, given a graph G, a set $S \subseteq V(G)$ is called an *odd cycle transversal* if $G \setminus S$ is a bipartite graph.

Odd Cycle Transversal (OCT)	**Parameter:** k
Input: An undirected multi-graph graph G, and an integer k.	
Question: Does G have an odd cycle transversal of size at most k?	

An instance of Odd Cycle Transversal is denoted by $\text{OCT}(G, k)$. The algorithm given by the result below shall be a central component in the design of our algorithm for BJB.

▶ **Proposition 3** ([26]). Odd Cycle Transversal *can be solved in time* $2.3146^k n^{\mathcal{O}(1)}$.

Apart from OCT, we also need to define an auxiliary problem that we call Annotated Bipartite-BJB (AB-BJB). As we proceed with our proofs, we shall continue defining auxiliary problems, where each problem captures a task more specific and technically more challenging than the previous one. The choice of this structure aims to ease the readability of our paper. Intuitively, AB-BJB is basically the BJB problem on bipartite graphs, with

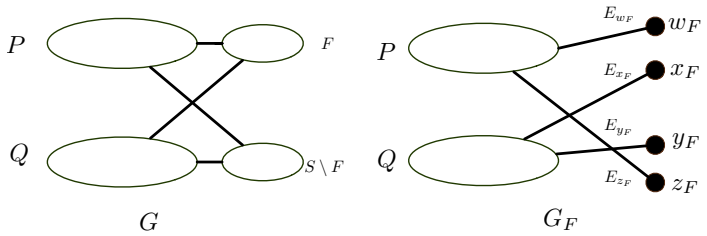


Figure 1 The construction in the proof of Theorem 1.

an extra constraint that demands that certain vertices are assigned a particular color by the witnessing coloring. We remark that the necessity of the reduction to bipartite graphs stems from the fact that we would like to employ Proposition 1 later. The formal definition of AB-BJB is given below.

Annotated Bipartite-BJB (AB-BJB)	**Parameter:** $k_1 + k_2$
Input: A bipartite multi-graph G with bipartition (P, Q), $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$, and integers μ, k_1 and k_2.	
Question: Does there exist a partition (V_1, V_2) of $V(G)$ such that $A \subseteq V_1$, $B \subseteq V_2$, $\lvert V_1\rvert = \mu$ and for $i \in \{1, 2\}$, $\lvert E(G[V_i])\rvert \leq k_i$?	

An instance of AB-BJB is denoted by AB-BJB(G, A, B, μ, k_1, k_2). A partition (V_1, V_2) satisfying the above properties is called a *witnessing coloring* of AB-BJB(G, A, B, μ, k_1, k_2). Furthermore, we need the following theorem, proven later in this paper.

▶ **Theorem 3.** AB-BJB *can be solved in time* $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$.

Let us now turn to focus on the proof of Theorem 1.

Proof of Theorem 1. Given an instance BJB(G, μ, k_1, k_2), call the algorithm given by Proposition 3 with the instance OCT(G, k) as input.

▶ **Claim 1.** *If* OCT(G, k) *is a* NO*-instance, then* BJB(G, μ, k_1, k_2) *is a* NO*-instance.*

Proof. Suppose BJB(G, μ, k_1, k_2) is a YES-instance. Let (V_1, V_2) be a witnessing coloring for this instance. Let $E' = E(G[V_1]) \cup E(G[V_2])$. Then, observe that $G \setminus E'$ is a bipartite graph. Let V' be a set of vertices of minimum size such that every edge in E' has at least one endpoint in V'. Since $|E'| \leq k$, it holds that $|V'| \leq k$. Moreover, $G \setminus V'$ is bipartite. Therefore, V' is an odd cycle transversal of G of size at most k. Thus, OCT(G, k) is a YES-instance. ◀

Henceforth, let S be an odd cycle transversal of G of size at most k. Then, $G \setminus S$ is a bipartite graph. Fix some bipartition (P, Q) of $G \setminus S$. Let $\mathcal{F}$ be the family of all subsets of S, that is, $\mathcal{F} = 2^S$. For any $F \in \mathcal{F}$, denote $l_1^F = |E(G[F])|$ and $l_2^F = |E(G[S \setminus F])|$, and let G_F be the graph constructed as follows (see Fig. 1).

- $V(G_F) = V(G \setminus S) \cup \{w_F, x_F, y_F, z_F\}$, where w_F, x_F, y_F, z_F are new distinct vertices.
- $E(G_F) = E(G \setminus S) \cup E_{w_F} \cup E_{x_F} \cup E_{y_F} \cup E_{z_F}$, where the *multisets* $E_{w_F}, E_{x_F}, E_{y_F}$ and E_{z_F} are defined as follows.
  - $E_{w_F} = \{e_u = (w_F, u) : u \in P,$ and there exists $v \in F$ such that $(u, v) \in E(G)\}$,
  - $E_{x_F} = \{e_u(x_F, u) : u \in Q,$ and there exists $v \in F$ such that $(u, v) \in E(G)\}$,
  - $E_{y_F} = \{e_u = (y_F, u) : u \in Q,$ and there exists $v \in S \setminus F$ such that $(u, v) \in E(G)\}$,
  - $E_{z_F} = \{e_u = (z_F, u) : u \in P,$ and there exists $v \in S \setminus F$ such that $(u, v) \in E(G)\}$.

Observe that G_F is a bipartite graph with $(P \cup \{x_F, y_F\}, Q \cup \{w_F, z_F\})$ as a bipartition.

▶ **Claim 2.** BJB(G, μ, k_1, k_2) *is a* YES*-instance if and only if there exists* $F \in \mathcal{F}$ *such that* AB-BJB$(G_F, \{w_F, x_F\}, \{y_F, z_F\}, \mu - |F| + 2, k_1 - l_1^F, k_2 - l_2^F)$ *is a* YES*-instance.*

Proof. In the forward direction, suppose that BJB(G, μ, k_1, k_2) is a YES-instance, and let (V_1, V_2) be a witnessing coloring for BJB(G, μ, k_1, k_2). Moreover, let $F = V_1 \cap S$. Now, we define a partition (V_1', V_2') of $V(G_F)$ as follows: $V_1' = (V_1 \setminus S) \cup \{x_F, y_F\}$ and $V_2' = (V_2 \setminus S) \cup \{w_F, z_F\}$. Let us now argue that (V_1', V_2') is a witnessing coloring for AB-BJB$(G_F, \{w_F, x_F\}, \{y_F, z_F\}, \mu - |F| + 2, k_1 - l_1^F, k_2 - l_2^F)$. First, by the construction of (V_1', V_2'), we have that $\{x_F, y_F\} \subseteq V_1'$ and $\{w_F, z_F\} \subseteq V_2'$. Second, as $V_1' = (V_1 \setminus S) \cup \{x_F, y_F\}$, we also have that $|V_1'| = |V_1| - |F| + 2 = \mu + |F| + 2$. Third, observe that for any $i \in \{1, 2\}$, $|E(G[V_i'])| = |E(G[V_i])| - |E(G[F])|$. Thus, $|E(G[V_i])| \leq k_i - l_i^F$.

In the backward direction, suppose that there exists an $F \in \mathcal{F}$ such that AB-BJB$(G_F, \{w_F, x_F\}, \{y_F, z_F\}, \mu - |F| + 2, k_1 - l_1^F, k_2 - l_2^F)$ is a YES-instance, and let (V_1', V_2') be a witnessing coloring for this instance. We now define a partition (V_1, V_2) of $V(G)$ as follows: $V_1 = (V_1' \cap V(G)) \cup F$ and $V_2 = (V_2' \cap V(G)) \cup (S \setminus F)$. Let us now argue that (V_1, V_2) is a witnessing coloring for BJB(G, μ, k_1, k_2). From the definition of V_1, and since $V(G) = (V(G_F) \setminus \{w_F, x_F, y_F, z_F\}) \cup F$ and $F \cap V(G_F) = \emptyset$, we have that $|V_1| = |V_1'| - |\{x_F, y_F\}| + |F| = \mu - |F| + 2 - 2 + |F| = \mu$. Moreover, observe that $|E(G[V_1])| = |E(G[V_1'])| + |E(G[F])| \leq k_1 + l_1^F$ and $|E(G[V_2])| = |E(G[V_2'])| + |E(G[S \setminus F])| \leq k_2 + l_2^F$. This concludes the proof of the claim. ◀

Thus, to solve an instance of BJB, it is enough to solve $2^{|S|} \leq 2^k$ instances of AB-BJB. Hence, by Theorem 3, BJB can be solved in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$. ◀

4 Solving Annotated Bipartite-BJB

Recall the problem definition of Annotated Bipartite-BJB (AB-BJB) from Section 3. In this section, we prove Theorem 3. For this purpose, let us define another auxiliary problem, which we call Annotated Bipartite Connected-BJB (ABC-BJB). Intuitively, ABC-BJB is exactly the same problem as AB-BJB where we are interested in an answer for every choice of $\mu \in [n]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, and additionally we demand the input graph to be connected.

Annotated Bipartite Connected-BJB (ABC-BJB) **Parameter:** $k_1 + k_2$

Input: A connected bipartite multi-graph $G = (P, Q)$, $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$, and integers k_1 and k_2.

Output: For all $\mu \in [n]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, output a binary value, $\texttt{aJP}[\mu, l_1, l_2]$, which is 1 if and only if there exists a partition (V_1, V_2) of $V(G)$ such that

- $A \subseteq V_1$ and $B \subseteq V_2$,
- $|V_1| = \mu$, and
- for $i \in \{1, 2\}$, $|E(G[V_i])| \leq l_i$.

For any $\mu \in [n]_0$, $l_1 \in [k_1]_0$, $l_2 \in [k_2]_0$, a partition witnessing that $\texttt{aJP}[\mu, l_1, l_2] = 1$ is called a *witnessing coloring* for $\texttt{aJP}[\mu, l_1, l_2] = 1$. Moreover, an instance of ABC-BJB is denoted by ABC-BJB(G, A, B, k_1, k_2). In the rest of this paper, we prove the following theorem.

▶ **Theorem 4.** ABC-BJB *can be solved in time* $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$.

Having Theorem 4 at hand, a simple application of the method of dynamic programming results in the proof of Theorem 3 (see full version of the paper [27]).

5 Solving Annotated Bipartite Connected-BJB

Recall the problem definition of ABC-BJB from Section 5. In this section, we prove Theorem 4. Let us start by stating a known result that is a crucial component of our proof. By this result, we would have an algorithm that efficiently computes a special type of tree decomposition, that we call a *highly connected tree decomposition*, where every bag is "highly-connected" rather than "small" as in the case of standard tree decompositions. While this property is the main feature of this decomposition, it is also equipped with other beneficial properties, such as a (non-trivial) upper bound on the size of its adhesions, which are *all* exploited by our algorithm.

▶ **Theorem 5** ([16]). *There exists an $2^{\mathcal{O}(k^2)}n^2m$-time algorithm that, given a* connected *graph G together with an integer k, computes a tree decomposition (T, β) of G with at most n nodes such that the following conditions hold, where $\eta = 2^{\mathcal{O}(k)}$.*

1. *For each $t \in V(T)$, the graph $G[\gamma(t) \setminus \sigma(t)]$ is connected and $N(\gamma(t) \setminus \sigma(t)) = \sigma(t)$.*
2. *For each $t \in V(T)$, the set $\beta(t)$ is (η, k)-unbreakable in $G[\gamma(t)]$.*
3. *For each non-root $t \in V(T)$, we have that $|\sigma(t)| \leq \eta$ and $\sigma(t)$ is $(2k, k)$-unbreakable in $G[\gamma(parent(t))]$.*

In order to process such a tree decomposition in a bottom-up fashion, relying on the method of dynamic programming, we need to address a specific problem associated with every bag, called Hypergraph Painting (HP). We chose the name HP to be consistent with the choice of problem name in [16], yet we stress that our problem is more general than the one in [16] (since the handling of a bag in our case is more intricate than the one in [16]).

Roughly speaking, an input of HP would consist of the following components. First, we are given "budget" parameters k_1 and k_2 as in an instance of ABC-BJB. Second, we are given an argument b which would simply be n (to upper bound $|\gamma(t)|$) when we construct an instance of HP while processing some node t in the tree decomposition. Third, we are given a hypergraph H which would essentially be the graph $G[\beta(t)]$ to which we add hyperedges that are supposed to represent the sets $\sigma(\widehat{t})$ for the children $\widehat{t}$ of t. Fourth, we are given an integer q whose purpose is clarified in the discussion below the definition of HP (in Definition 8). Finally, for every hyperedge F, we are given a function $f_F : [k]_0^F \times [b]_0 \times [k_1]_0 \times [k_2]_0 \to \{0, 1\}$. To roughly understand the meaning of this function, first recall that F is supposed to represent $\sigma(\widehat{t})$ for some child $\widehat{t}$ of t. Now, the function f_F aims to capture all information obtained while we processed the child $\widehat{t}$ of t that might be relevant to the node t. In particular, let us give an informal, intuitive interpretation of an element (Γ, μ, l_1, l_2) in the domain of f_F. For this purpose, note that when we remove at most k edges from the (connected) graph $G[\gamma(\widehat{t})]$, we obtain at most $k + 1$ connected components. The function Γ can be thought of as a method to assign to each vertex in $\sigma(\widehat{t})$ the connected component in which it should lie. Such information is extremely useful since each such connected component is in particular a bipartite graph, and hence by relying on Proposition 1 and an exhaustive search, we would be able to use it to extract a witnessing coloring for an instance of ABC-BJB. The arguments μ, l_1 and l_2 can be thought of as those in the definition of an output of ABC-BJB. Now, the value $f_F(\Gamma, \mu, l_1, l_2)$ aims to indicate whether Γ, μ, l_1 and l_2 are "realizable" in the context of the child $\widehat{t}$ (the precise meaning of this value would become clearer later, once we establish additional necessary definitions.)

Let us now give the formal definition of HP. In this definition, we denote $k = k_1 + k_2$.

Hypergraph Painting (HP)
Input: Integers k_1, k_2, b, d and q, a multi-hypergraph H with hyperedges of size at most d, and for all $F \in E(H)$, a function $f_F : [k]_0^F \times [b]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$.
Output: For all $0 \leq \mu \leq b$, $0 \leq l_1 \leq k_1$, $0 \leq l_2 \leq k_2$, output the binary value

$$\texttt{aHP}[\mu, l_1, l_2] = \bigvee_{\Upsilon : V(H) \to [k]_0} \bigvee_{\substack{\{\mu^F\}|_{F \in E(H)} \\ \{l_1^F\}|_{F \in E(H)} \\ \{l_2^F\}|_{F \in E(H)}}} \bigwedge_{F \in E(H)} f_F(\Upsilon|_F, \mu^F, l_1^F, l_2^F),$$

where $\mu = \sum_{F \in E(H)} \mu^F$, $\sum_{F \in E(H)} l_1^F \leq l_1$, $\sum_{F \in E(H)} l_2^F \leq l_2$ and each of μ^F, l_1^F and l_2^F is a non-negative integer.

For a particular choice of μ, l_1 and l_2, a function Υ witnessing that $\texttt{aHP}[\mu, l_1, l_2] = 1$ is called a witnessing coloring for $\texttt{aHP}[\mu, l_1, l_2]$. An instance of Hypergraph Painting is denoted by $\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$.

Although we are not able to tackle HP efficiently at its full generality, we are still able to solve those instances that are constructed when we would like to "handle" a single bag in a highly connected tree decomposition. For the sake of clarity, let us now address the beneficial properties that these instances satisfy individually, where each of them ultimately aims to ease our search for a witnessing coloring. The first property, called local unbreakability, unconditionally restricts the way a function $\Gamma : F \to [k]_0$, to be thought of as a restriction of the witnessing coloring we seek, can color a hyperedge F so that the value of f_F is 1.[3]

▶ **Definition 6** (Local Unbreakability)**.** An instance $\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ is *locally unbreakable* if every $F \in E(H)$ satisfies the following property: for any $\Gamma : F \to [k]_0$ that is not $(3k^2, k)$-unbreakable, $f_F(\Gamma, \mu, l_1, l_2) = 0$ for all $0 \leq \mu \leq b$, $0 \leq l_1 \leq k_1$ and $0 \leq l_2 \leq k_2$.

The second property, called connectivity, implies that if we would like to use a function $\Gamma : F \to [k]_0$ to color a hyperedge (as a restriction of a witnessing coloring) with more than one color, then we would have to "pay" at least 1 from our budget $l_1 + l_2$.

▶ **Definition 7** (Connectivity)**.** An instance $\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ is *connected* if every $F \in E(H)$ satisfies the following property: for any $\Gamma : F \to [k]_0$ for which there exist distinct $i, j \in [k]_0$ such that $|\Gamma^{-1}(i)|, |\Gamma^{-1}(j)| > 0$, it holds that $f_F(\Gamma, \mu, l_1, l_2) = 1$ only if $l_1 + l_2 \geq 1$.

The third property, called global unbreakability, directly restricts our "solution space" by implying that we only need to determine whether there exists a (q, k)-unbreakable witnessing coloring.

▶ **Definition 8** (Global Unbreakability)**.** An instance $\mathrm{HP}(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ is *globally unbreakable* if for all $0 \leq \mu \leq b$, $0 \leq l_1 \leq k_1$, $0 \leq l_2 \leq k_2$: if $\texttt{aHP}[\mu, l_1, l_2] = 1$, then there exists a witnessing coloring $\Upsilon : V(H) \to [k]_0$ that is (q, k)-unbreakable.

An instance $HP(k_1, k_2, b, d, q, H, \{f_F\}|_{F \in E(H)})$ is called a *favorable instance of* HP if it is locally unbreakable, connected and globally unbreakable. For such instances we have the following theorem.

[3] In this context, it may be insightful to recall Lemma 2.

▶ **Theorem 9** (⋆).
HP *on favorable instances is solvable in time* $2^{\mathcal{O}(\min(k,q)\log(k+q))}d^{\mathcal{O}(k^2)}m^{\mathcal{O}(1)}$.

The proof of this theorem is very technical, involving non-trivial analysis of a very "messy" picture obtained by guessing part of a hypothetical witnessing coloring via the method of color coding. Due to space constraints, the details are omitted here but can we found in the full version of the paper [27].

From now onwards, to simplify the presentation of arguments ahead with respect to ABC-BJB, we would abuse notation and directly define a witnessing coloring as a function rather than a partition. More precisely, the term witnessing coloring for $\texttt{aJP}[\mu, l_1, l_2] = 1$ would refer to a function $col : V(G) \to \{V_1, V_2\}$ such that $A \subseteq V_1$, $B \subseteq V_2$, $|V_1| = \mu$ and for $i \in \{1, 2\}$, $|E(G[V_i])| \leq l_i$. To proceed to our proof of Theorem 4, we first need to introduce an additional notation. Roughly speaking, this notation translates a coloring Υ of the form that witnesses some $\texttt{aHP}[\mu, l_1, l_2] = 1$ to a coloring of the form that witnesses $\texttt{aJP}[\mu, l_1, l_2] = 1$ via some tuple $\mathbf{v} \in \{0, 1\}^{k+1}$. Formally,

▶ **Definition 10.** For a tuple $\mathbf{v} \in \{0, 1\}^{k+1}$, bipartite graph G with bipartition (P, Q), $A \subseteq V(G)$ and $\Upsilon : A \to [k]_0$, define $\widehat{\Upsilon}_{\mathbf{v}} : A \to \{V_1, V_2\}$ as follows.

- For all $v \in P$, $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_1$ if and only if $\mathbf{v}[\Upsilon(v)] = 0$.
- For all $v \in Q$, $\widehat{\Upsilon}_{\mathbf{v}}(v) = V_1$ if and only if $\mathbf{v}[\Upsilon(v)] = 1$.

Suppose we are given an instance ABC-BJB(G, A, B, k_1, k_2). Fix some bipartition (P, Q) of G. Let (T, β) be the highly connected tree decomposition computed by the algorithm of Theorem 5, and let r be the root of T. In what follows, $\eta = 2^{\mathcal{O}(k)}$ as in Theorem 5, and $q = (\eta + k)k$. We now proceed to define a binary variable that is supposed to represent the answer we would like to compute when we process the bag of a specific node of the tree. Hence, one of the arguments is a node t, and three additional arguments are $\mu \in [n]_0, l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$. However, we cannot be satisfied with one answer, but need an answer for every possible "interaction" between the bag of t and the bag of its parent t'. Thus, the definition also includes a coloring of $\sigma(t)$. The tuple $\mathbf{v} \in \{0, 1\}^{k+1}$ is necessary for the translation process described in Definition 10 (the way in which we shall obtain such a "right" tuple later in the proof would essentially rely on brute-force).

▶ **Definition 11.** Given $t \in V(T)$, a $(3k^2, k)$-unbreakable function $\Upsilon^\sigma : \sigma(t) \to [k]_0$, a tuple $\mathbf{v} \in \{0, 1\}^{k+1}$, and integers $\mu \in [n]_0, l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, the binary variable $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ is 1 if and only if there exists $\Upsilon : \gamma(t) \to [k]_0$ extending Υ^σ such that

1. The translation $\widehat{\Upsilon}_{\mathbf{v}}$ maps to V_1 exactly μ vertices, that is, $|\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)| = \mu$.
2. The translation $\widehat{\Upsilon}_{\mathbf{v}}$ maps $A \cap \gamma(t)$ to V_1 and $B \cap \gamma(t)$ to V_2, that is, $A \cap \gamma(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)$ and $B \cap \gamma(t) \subseteq \widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)$.
3. For all $i \in \{1, 2\}$, it holds that $|E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_i)])| \leq l_i$.
4. The set of edges between vertices receiving different colors by Υ is exactly the set of edges between vertices that are mapped to the same side by the translation $\widehat{\Upsilon}_{\mathbf{v}}$, that is,

$$\bigcup_{i,j \in [k]_0, i \neq j} E(\Upsilon^{-1}(i), \Upsilon^{-1}(j)) = E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_1)]) \cup E(G[\widehat{\Upsilon}_{\mathbf{v}}^{-1}(V_2)]).$$

A function Υ as above is called a *witnessing coloring* for $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$.

▶ **Lemma 12** (⋆). *For any $\mu \in [n]_0$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, $\texttt{aJP}[\mu, l_1, l_2] = 1$ if and only if there exists $\mathbf{v} \in \{0, 1\}^{k+1}$ such that $y[r, \phi, \mathbf{v}, \mu, l_1, l_2] = 1$.*

By Lemma 12, it is sufficient to compute $y[r, \phi, \mathbf{v}, \mu, l_1, l_2]$ for all $\mu \in [n]$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$. To this end, we need to compute $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ for every node $t \in V(T)$, function $\Upsilon^\sigma : \sigma(t) \to [k]_0$ that is $(3k^2, k)$-unbreakable, tuple $\mathbf{v} \in \{0,1\}^{k+1}$, and integers $\mu \in [n]_0, l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$. Here, we employ bottom-up dynamic programming over the tree decomposition (T, β). Let us now zoom into the computation of $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ for all $\mu \in [n]$, $l_1 \in [k_1]_0$ and $l_2 \in [k_2]_0$, for some specific t, Υ^σ and $\mathbf{v}$. Note that we now assume that values corresponding to the children of t (if such children exist) have been already computed correctly. Moreover, note that $|\sigma(t)| \leq \eta$, the number of $(3k^2, k)$-unbreakable functions $\Upsilon^\sigma : \sigma(t) \to [k]_0$ is at most $|\eta|^{k^{\mathcal{O}(1)}} = 2^{k^{\mathcal{O}(1)}}$ (by Lemma 2), and the number of binary vectors of size $k+1$ is at most 2^{k+1}. Thus, the total running time would consist of the computation time of (T, β), and $n \cdot q^{\mathcal{O}(k)} \cdot 2^{k+1}$ times the computation time for a set of values as the one we examine now. Hence, it remains to show how to compute the current set of values in time $2^{k^{\mathcal{O}(1)}}$.

To compute our current set of values, let us construct an instance $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$ of HP where $V(H) = \beta(t)$, and $E(H)$ and $\{f_F\}|_{F \in E(H)}$ are defined as follows.

1. **Type-1 Hyperedges.** For all $v \in \beta(t)$, insert $F = \{v\}$ into $E(H)$. Define $f_F : [k]_0^F \times [n]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$ as

$$f_F(\Gamma, \mu, l_1, l_2) = \begin{cases} 0, & \text{if } v \in \sigma(t) \text{ and } \Gamma(v) \neq \Upsilon^\sigma(v) \\ 1, & \text{if } v \in A,\ \widehat{\Gamma}_{\mathbf{v}}(F) = V_1,\ l_1 = l_2 = 0 \text{ and } \mu = 1 \\ 1, & \text{if } v \in B,\ \widehat{\Gamma}_{\mathbf{v}}(F) = V_2,\ l_1 = l_2 = 0 \text{ and } \mu = 0 \\ 1, & \text{if } v \notin A \cup B,\ l_1 = l_2 = 0 \text{ and } \mu = [\widehat{\Gamma}_{\mathbf{v}}(F) = V_1] \\ 0, & \text{otherwise} \end{cases}$$

2. **Type-2 Hyperedges.** For all $(u, v) \in E(G[\beta(t)])$, add $F = \{u, v\}$ in $E(H)$. Define $f_F : [k]_0^F \times [n]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$ as

$$f_F(\Gamma, \mu, l_1, l_2) = \begin{cases} 0, & \text{if } \mu \neq 0 \\ 1, & \text{if } \widehat{\Gamma}_{\mathbf{v}}(u) \neq \widehat{\Gamma}_{\mathbf{v}}(v) \text{ and } \Gamma(u) = \Gamma(v) \\ 1, & \text{if } \widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v) = V_1 \text{ and } l_1 \geq 1 \\ 1, & \text{if } \widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v) = V_2 \text{ and } l_2 \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

3. **Type-3 Hyperedges.** For all $\widehat{t} \in V(T)$ that is a child of t in the tree T, insert $F = \sigma(\widehat{t})$ into $E(H)$. Define $f_F : [k]_0^F \times [n]_0 \times [k_1]_0 \times [k_2]_0 \to \{0,1\}$ as

$$f_F(\Gamma, \mu, l_1, l_2) = \begin{cases} 0, & \text{if } \Gamma \text{ is not } (3k^2, k)\text{-unbreakable or } y[\widehat{t}, \Gamma, \mu + \mu', l_1 + l_1', l_2 + l_2'] = 0 \\ 1, & \text{otherwise} \end{cases}$$

where $\mu' = |\widehat{\Gamma}_{\mathbf{v}}^{-1}(V_1)|$, and $l_i' = |\{\{u, v\} \in E(G[\sigma(\widehat{t})]) : \widehat{\Gamma}_{\mathbf{v}}(u) = \widehat{\Gamma}_{\mathbf{v}}(v) = V_i\}|$ for $i \in [2]$.

Let us first claim that witnessing colorings related to $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F \in E(H)})$ are useful in the sense that they can be extended to witnessing colorings for the binary values in which we are interested.

▶ **Lemma 13** (⋆). *For all $\mu \in [n]$, $l_1 \in [k_1]_0$, $l_2 \in [k_2]_0$, if $\mathtt{aHP}[\mu, l_1, l_2] = 1$, then $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$. In fact, for any witness $\Upsilon : \beta(t) \to [k]_0$ of $\mathtt{aHP}[\mu, l_1, l_2] = 1$, there exists a function $\Upsilon' : \gamma(t) \to [k]_0$ that extends Υ and witnesses $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$.*

In light of Lemma 13, we now turn to verify that $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F\in E(H)})$ is of the form that we are actually able to solve.

▶ **Lemma 14** (⋆). $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F\in E(H)})$ *is a favorable instance of* HP.

Finally, we turn to address the statement complementary to the one of Lemma 13.

▶ **Lemma 15** (⋆). *For all* $\mu \in [n]$, $l_1 \in [k_1]_0$ *and* $l_2 \in [k_2]_0$, *if* $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2] = 1$, *then* $\mathit{aHP}[\mu, l_1, l_2] = 1$.

Recall that we have argued that to prove Theorem 9, it is sufficient to show that the current set of values $y[t, \Upsilon^\sigma, \mathbf{v}, \mu, l_1, l_2]$ can be computed in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$. Here, n refers to $|V(G)|$. By Lemmas 13 and 15, this set of values can be derived from the solution of $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F\in E(H)})$. Since $\mathrm{HP}(k_1, k_2, n, \eta, q, H, \{f_F\}|_{F\in E(H)})$ is a favorable instance of HP (by Lemma 14), the algorithm given by Theorem 9 solves it in time $2^{\mathcal{O}(\min(k,q)\log(k+q))} d^{\mathcal{O}(k^2)} |E(H)|^{\mathcal{O}(1)} = 2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$. This concludes the proof of Theorem 9.

References

1. Noga Alon, Béla Bollobás, Michael Krivelevich, and Benny Sudakov. Maximum cuts and judicious partitions in graphs without short cycles. *Journal of Combinatorial Theory, Series B*, 88(2):329–346, 2003.
2. Florian Barbero, Gregory Gutin, Mark Jones, and Bin Sheng. Parameterized and approximation algorithms for the load coloring problem. *Algorithmica*, pages 1–19, 2015.
3. Béla Bollobás and Alex D Scott. Judicious partitions of graphs. *Periodica Mathematica Hungarica*, 26(2):125–137, 1993.
4. Béla Bollobás and Alex D Scott. Judicious partitions of hypergraphs. *Journal of Combinatorial Theory, Series A*, 78(1):15–31, 1997.
5. Béla Bollobás and Alex D Scott. Exact bounds for judicious partitions of graphs. *Combinatorica*, 19(4):473–486, 1999.
6. Béla Bollobás and Alex D Scott. Judicious partitions of 3-uniform hypergraphs. *European Journal of Combinatorics*, 21(3):289–300, 2000.
7. Béla Bollobás and Alex D Scott. Problems and results on judicious partitions. *Random Structures & Algorithms*, 21(3-4):414–430, 2002.
8. Béla Bollobás and Alex D Scott. Judicious partitions of bounded-degree graphs. *Journal of Graph Theory*, 46(2):131–143, 2004.
9. Béla Bollobás and Alex D Scott. Max k-cut and judicious k-partitions. *Discrete Mathematics*, 310(15):2126–2139, 2010.
10. Edouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Multi-parameter analysis for local graph partitioning problems: Using greediness for parameterization. *Algorithmica*, 71(3):566–580, 2015. doi:10.1007/s00453-014-9920-6.
11. Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *Comput. J.*, 51(1):102–121, 2008. doi:10.1093/comjnl/bxm086.
12. Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.
13. Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016.
14. Robert Crowston, Gregory Gutin, Mark Jones, and Gabriele Muciaccia. Maximum balanced subgraph problem parameterized above lower bound. *Theor. Comput. Sci.*, 513:53–64, 2013.
15. Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the edwards-erdős bound. *Algorithmica*, 72(3):734–757, 2015.

16 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 323–332. ACM, 2014. doi:10.1145/2591796.2591852.

17 Christopher S Edwards. Some extremal properties of bipartite subgraphs. *Canad. J. Math*, 25(3):475–483, 1973.

18 Christopher S Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Proc. 2nd Czechoslovak Symposium on Graph Theory, Prague*, pages 167–181, 1975.

19 Michael Etscheid and Matthias Mnich. Linear kernels and linear-time algorithms for finding large cuts. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, volume 64 of *LIPIcs*, pages 31:1–31:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.31.

20 Gregory Gutin and Mark Jones. Parameterized algorithms for load coloring problem. *Information Processing Letters*, 114(8):446–449, 2014.

21 Gregory Gutin and Anders Yeo. Note on maximal bisection above tight lower bound. *Information Processing Letters*, 110(21):966–969, 2010.

22 Gregory Gutin and Anders Yeo. Constraint satisfaction problems parameterized above or below tight bounds: A survey. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 257–286. Springer, 2012.

23 John Haslegrave. Judicious partitions of uniform hypergraphs. *Combinatorica*, 34(5):561–572, 2014.

24 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169, 2011.

25 Choongbum Lee, Po-Shen Loh, and Benny Sudakov. Judicious partitions of directed graphs. *Random Structures & Algorithms*, 48(1):147–170, 2016.

26 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

27 Daniel Lokshtanov, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Balanced judicious partition is fixed-parameter tractable. *arXiv preprint arXiv:1710.05491*, 2017.

28 Jie Ma and Xingxing Yu. On judicious bipartitions of graphs. *Combinatorica*, 36(5):537–556, 2016.

29 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.

30 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *J. Comput. Syst. Sci.*, 75(2):137–153, 2009.

31 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

32 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014.

33 Matthias Mnich and Rico Zenklusen. Bisections above tight lower bounds. In *Graph-Theoretic Concepts in Computer Science - 38th International Workshop, WG 2012, Jerusalem, Israel, June 26-28, 2012, Revised Selcted Papers*, volume 7551 of *Lecture Notes in Computer Science*, pages 184–193. Springer, 2012.

34 Saket Saurabh and Meirav Zehavi. $(k, n-k)$-max-cut: An $\mathcal{O}^*(2^p)$-time algorithm and a polynomial kernel. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors,

LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings, volume 9644 of *Lecture Notes in Computer Science*, pages 686–699. Springer, 2016. doi:10.1007/978-3-662-49529-2_51.

35 Alex D Scott. Judicious partitions and related problems. *Surveys in combinatorics*, 327:95–117, 2005.

36 Hadas Shachnai and Meirav Zehavi. Representative families: A unified tradeoff-based approach. *J. Comput. Syst. Sci.*, 82(3):488–502, 2016. doi:10.1016/j.jcss.2015.11.008.

37 Baogang Xu, Juan Yan, and Xingxing Yu. Balanced judicious bipartitions of graphs. *Journal of Graph Theory*, 63(3):210–225, 2010.

38 Baogang Xu and Xingxing Yu. Judicious k-partitions of graphs. *Journal of Combinatorial Theory, Series B*, 99(2):324–337, 2009.

39 Baogang Xu and Xingxing Yu. On judicious bisections of graphs. *J. Comb. Theory, Ser. B*, 106:30–69, 2014.

On Hashing-Based Approaches to Approximate DNF-Counting*†‡

Kuldeep S. Meel[§1], Aditya A. Shrotri[2], and Moshe Y. Vardi[3]

1 **National University of Singapore, Singapore, Singapore**
meel@comp.nus.edu.sg
2 **Rice University, Houston, USA**
as128@rice.edu
3 **Rice University, Houston, USA**
vardi@rice.edu

Abstract

Propositional model counting is a fundamental problem in artificial intelligence with a wide variety of applications, such as probabilistic inference, decision making under uncertainty, and probabilistic databases. Consequently, the problem is of theoretical as well as practical interest. When the constraints are expressed as DNF formulas, Monte Carlo-based techniques have been shown to provide a fully polynomial randomized approximation scheme (FPRAS). For CNF constraints, hashing-based approximation techniques have been demonstrated to be highly successful. Furthermore, it was shown that hashing-based techniques also yield an FPRAS for DNF counting without usage of Monte Carlo sampling. Our analysis, however, shows that the proposed hashing-based approach to DNF counting provides poor time complexity compared to the Monte Carlo-based DNF counting techniques. Given the success of hashing-based techniques for CNF constraints, it is natural to ask: Can hashing-based techniques provide an efficient FPRAS for DNF counting? In this paper, we provide a positive answer to this question. To this end, we introduce two novel algorithmic techniques: *Symbolic Hashing* and *Stochastic Cell Counting*, along with a new hash family of *Row-Echelon hash functions*. These innovations allow us to design a hashing-based FPRAS for DNF counting of similar complexity (up to polylog factors) as that of prior works. Furthermore, we expect these techniques to have potential applications beyond DNF counting.

1998 ACM Subject Classification G.1.2 Special Function Approximation, F.4.1 Logic and Constraint Programming

Keywords and phrases Model Counting, Approximation, DNF, Hash Functions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.41

1 Introduction

Propositional model counting is a fundamental problem in artificial intelligence with a wide range of applications including probabilistic inference, databases, decision making under

* The author list has been sorted alphabetically by last name; this should not be used to determine the extent of authors' contributions.
† The full version of this paper is available at https://arxiv.org/abs/1710.05247.
‡ Work supported in part by NSF grants CCF-1319459 and IIS-1527668 and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering".
§ Kuldeep S. Meel is supported by the IBM PhD Fellowship and the Lodieska Stockbridge Vaughn Fellowship.

uncertainty, and the like [1, 6, 7, 23]. Given a Boolean formula ϕ, the problem of propositional model counting , also referred to as #SAT, is to compute the number of solutions of ϕ [28]. Depending on whether ϕ is expressed as a CNF or DNF formula, the corresponding model counting problems are denoted as #CNF or #DNF, respectively. Both #CNF and #DNF have a wide variety of applications. For example, probabilistic-inference queries reduce to solving #CNF instances [1, 20, 21, 23], while evaluation of queries for probabilistic database reduce to #DNF instances [6]. Consequently, both #CNF and #DNF have been of theoretical as well as practical interest over the years [16, 18, 22, 24]. In his seminal paper, Valiant [28] showed that both #CNF and #DNF are #P-complete, a class of problems that are believed to be intractable in general.

Given the intractability of #CNF and #DNF, much of the interest lies in the approximate variants of #CNF and #DNF, wherein for given tolerance and confidence parameters ε and δ, the goal is to compute an estimate C such that C is within a $(1+\varepsilon)$ multiplicative factor of the true count with confidence at least $1-\delta$. While both #CNF and #DNF are #P-complete in their exact forms, the approximate variants differ in complexity: approximating #DNF can be accomplished in fully polynomial randomized time [5, 17, 18], but approximate #CNF is NP-hard [24]. Consequently, different techniques have emerged to design scalable approximation techniques for #DNF and #CNF.

In the context of #DNF, the works of Karp, Luby, and Madras [17, 18] led to the development of highly efficient Monte-Carlo based techniques, whose time complexity is linear in the size of the formula. On the other hand, hashing-based techniques have emerged as a scalable approach to the approximate model counting of CNF formulas [3, 4, 10, 13, 24], and are effective even for problems with existing FPRAS such as *network reliability* [8]. These hashing-based techniques employ 2-universal hash functions to partition the space of satisfying solutions of a CNF formula into cells such that a randomly chosen cell contains only a *s*mall number of solutions. Furthermore, it is shown that the number of solutions across the cells is *roughly equal* and, therefore, an estimate of the total count can be obtained by counting the number of solutions in a cell and scaling the obtained count by the number of cells. Since the problem of counting the number of solutions in a cell when the number of solutions is *small* can be accomplished efficiently by invoking a SAT solver, the hashing-based techniques can take advantage of the recent progress in the development of efficient SAT solvers. Consequently, algorithms such as ApproxMC [3, 4] have been shown to scale to instances with hundreds of thousands of variables.

While Monte Carlo techniques introduced in the works of Karp et al. have shown to not be applicable in the context of approximate #CNF [18], it was not known whether hashing-based techniques could be employed to obtain efficient algorithms for #DNF. Recently, significant progress in this direction was achieved by Chakraborty, Meel and Vardi [4], who showed that hashing-based framework of ApproxMC could be employed to obtain FPRAS for #DNF counting[1]. There is, however, no precise complexity analysis in [4]. In this paper, we provide a complexity analysis of the proposed scheme of Chakraborty et al., which is worse than quartic in the size of formula. In comparison, state-of- the-art approaches achieve complexity linear in the number of variables and cubes for #DNF counting. This begs the question: *How powerful is the hashing-based framework in the context of DNF counting? In particular, can it lead to algorithms competitive in runtime complexity with state-of-the-art?*

[1] It is worth noting that several hashing-based algorithms based on [10, 27] do not lead to FPRAS schemes for #DNF despite close similarity to Chakraborty et al.'s approach

In this paper, we provide a positive answer to this question. To achieve such a significant reduction in complexity, we offer three novel algorithmic techniques: (i) A new class of 2-universal hash functions that enable fast enumeration of solutions using Gray Codes, (ii) Symbolic Hashing, and (iii) Stochastic Cell Counting. These techniques allow us to achieve the complexity of $\tilde{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$, which is within polylog factors of the complexity achieved by Karp et al. [18]. Here, m and n are the number of cubes and variables respectively while ε and δ are the tolerance and confidence of approximation. Furthermore, we believe that these techniques are not restricted to #DNF. Given recent breakthroughs achieved in the development of hashing-based CNF-counting techniques, we believe our techniques have the potential for a wide variety of applications.

The rest of the paper is organized as follows: we introduce notation in section 2 and discuss related work in section 3. We describe our main contributions in section 4, analyze the resulting algorithm in section 5 and discuss future work and conclude in section 6.

2 Preliminaries

DNF Formulas and Counting

We use Greek letters ϕ, θ and ψ to denote boolean formulas. A formula ϕ over boolean variables $x_1, x_2, \ldots, x_{\mathsf{n}}$ is in Disjunctive Normal Form (DNF) if it is a disjunction over conjunctions of variables or their negations. We use X to denote the set of variables appearing in the formula. Each occurrence of a variable or its negation is called a *literal*. Disjuncts in the formula are called *cubes* and we denote the i^{th} cube by ϕ^{Ci}. Thus $\phi = \phi^{C1} \vee \phi^{C2} \vee \ldots \vee \phi^{C\mathsf{m}}$ where each ϕ^{Ci} is a conjunction of *literals*. We will use n and m to denote the number of variables and number of cubes in the input DNF formula, respectively. The number of literals in a cube ϕ^{Ci} is called its *width* and is denoted by $width[\phi^{Ci}]$.

An assignment to all the variables can be represented by a vector $\boldsymbol{x} \in \{0,1\}^{\mathsf{n}}$ with 1 corresponding to $true$ and 0 to $false$. $U = \{0,1\}^{\mathsf{n}}$ is the set of all possible assignments, which we refer to as the *universe* or state space interchangeably. An assignment $\boldsymbol{x}$ is called a satisfying assignment for a formula ϕ if ϕ evaluates to $true$ under $\boldsymbol{x}$. In other words $\boldsymbol{x}$ satisfies ϕ and is denoted as $\boldsymbol{x} \models \phi$. Note that an assignment $\boldsymbol{x}$ will satisfy a DNF formula ϕ if $\boldsymbol{x} \models \phi^{Ci}$ for some i. The DNF-Counting Problem is to count the number of satisfying assignments of a DNF formula.

Next, we formalize the concept of a counting problem. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation which is decidable in polynomial time and there is a polynomial p such that for every $(s,t) \in R$ we have $|t| \leq p(|s|)$. The decision problem corresponding to R asks if for a given s there exists a t such that $(s,t) \in R$. Such a problem is in NP. Here, s is a called the *problem instance* and t is called the *witness*. We denote the set of all witnesses for a given s by R_s. The *counting problem* corresponding to R is to calculate the size of the witness set $|R_s|$ for a given s. Such a problem is in #P[28]. The DNF-Counting problem is an example of this formalism: A formula ϕ is a problem instance and a satisfying assignment $\boldsymbol{x}$ is a witness of ϕ. The set of satisfying assignments or the solution space is denoted R_ϕ and the goal is to compute $|R_\phi|$. It is known that the problem is #P-Complete, which is believed to be intractable [26]. Therefore, we look at what it means to efficiently and accurately approximate this problem.

A *fully polynomial randomized approximation scheme* (FPRAS) is a randomized algorithm that takes as input a problem instance s, a tolerance $\varepsilon \in (0,1)$ and confidence parameter $\delta \in (0,1)$ and outputs a random variable c such that $\Pr[\frac{1}{1+\varepsilon}|R_s| \leq c \leq (1+\varepsilon)|R_s|] \geq 1-\delta$ and the running time of the algorithm is polynomial in $|s|$, $1/\varepsilon$, $\log(1/\delta)$ [17]. Notably,

while exact DNF-counting is inter-reducible with exact CNF-counting, the approximate versions of the two problems are not because multiplicative approximation is not closed under complementation.

Matrix Notation

We use x,y, z,... to denote scalar variables. We use subscripts $x_1, x_2, \ldots$ as required. In this paper we are dealing with operations over the boolean ring, where the variables are boolean, 'addition' is the XOR operation ($\oplus$) and 'multiplication' is the AND operation ($\wedge$). We use the letters i,j,k,l as indices or to denote positions. We denote sets by non-boldface capital letters. We use capital boldface letters $\boldsymbol{A}$,$\boldsymbol{B}$,... to denote matrices, small boldface letters $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$,... to denote vectors. $\boldsymbol{A}^{[\mathsf{p}\times\mathsf{q}]}$ denotes a matrix of p rows and q columns, while $\boldsymbol{u}^{[\mathsf{q}]}$ denotes a vector of length q. $\boldsymbol{0}^{[\mathsf{q}]}$ and $\boldsymbol{1}^{[\mathsf{q}]}$ are the all 0s and all 1s vectors of length n, respectively. We omit the dimensions when clear from context. $\boldsymbol{x}[i]$ denotes the i^{th} element of $\boldsymbol{x}$, while $\boldsymbol{A}[i,j]$ denotes the element in the ith row and j^{th} column of $\boldsymbol{A}$. $\boldsymbol{A}[r_1 : r_2, c_1 : c_2]$ denotes the sub-matrix of $\boldsymbol{A}$ between rows r_1 and r_2 excluding r_2 and columns c_1 and c_2 excluding c_2. Similarly $\boldsymbol{v}[i : j]$ denotes the sub-vector of $\boldsymbol{v}$ between index i and index j excluding j. The i^{th} row of $\boldsymbol{A}$ is denoted $\boldsymbol{A}[i,:]$ and j^{th} column as $\boldsymbol{A}[:,j]$. The $\mathsf{p}\times(\mathsf{q}_1+\mathsf{q}_2)$ matrix formed by concatenating rows of matrices $\boldsymbol{A}^{[\mathsf{p}\times\mathsf{q}_1]}$ and $\boldsymbol{B}^{[\mathsf{p}\times\mathsf{q}_2]}$ is written in block notation as $[\boldsymbol{A} \mid \boldsymbol{B}]$, while $[\frac{\boldsymbol{A}}{\boldsymbol{B}}]$ represents concatenation of columns. Similarly the $(\mathsf{q}_1+\mathsf{q}_2)$-length concatenation of vectors $\boldsymbol{v}^{[\mathsf{q}_1]}$ and $\boldsymbol{w}^{[\mathsf{q}_2]}$ is $[\boldsymbol{v} \mid \boldsymbol{w}]$. The dot product between matrix $\boldsymbol{A}$ and vector $\boldsymbol{x}$ is written as $\boldsymbol{A}.\boldsymbol{x}$. The vector formed by element-wise XOR of vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ is denoted $\boldsymbol{v}\oplus\boldsymbol{w}$.

Hash Functions

A hash function $h : \{0,1\}^{\mathsf{q}} \to \{0,1\}^{\mathsf{p}}$ partitions the elements of of the domain $\{0,1\}^{\mathsf{q}}$ into 2^{p} cells. $h(\boldsymbol{x}) = \boldsymbol{y}$ implies that h maps the assignment $\boldsymbol{x}$ to the cell $\boldsymbol{y}$. $h^{-1}(\boldsymbol{y}) = \{\boldsymbol{x} | h(\boldsymbol{x}) = \boldsymbol{y}\}$ is the set of assignments that map to the cell $\boldsymbol{y}$. In the context of counting, 2-universal families of hash functions, denoted by $H(\mathsf{q},\mathsf{p},2)$, are of particular importance. When h is sampled uniformly at random from $H(\mathsf{q},\mathsf{p},2)$, 2-universality entails

1. $\Pr[h(\boldsymbol{x}_1) = h(\boldsymbol{x}_2)] \leq 2^{-\mathsf{p}}$ for all $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$
2. $\Pr[h(\boldsymbol{x}) = \boldsymbol{y}] = 2^{-\mathsf{p}}$ for every $\boldsymbol{x} \in \{0,1\}^{\mathsf{q}}$ and $\boldsymbol{y} \in \{0,1\}^{\mathsf{p}}$.

Of particular interest is the random XOR family of hash functions, which is defined as $H_{XOR}(\mathsf{q},\mathsf{p}) = \{\boldsymbol{A}.\boldsymbol{x}\oplus\boldsymbol{b} \mid \boldsymbol{A}[i,j] \in \{0,1\}$ *and* $\boldsymbol{b}[i] \in \{0,1\}\ ;\ 0 \leq i < \mathsf{p},\ 0 \leq j < \mathsf{q}\}$. Selecting $\boldsymbol{A}[i,j]s$ and $\boldsymbol{b}[i]$s randomly from $\{0,1\}$ is equivalent to drawing uniformly at random from this family. A pair $\boldsymbol{A}$ and $\boldsymbol{b}$ now defines a hash function $h_{\boldsymbol{A},\boldsymbol{b}}$ as follows: $h_{\boldsymbol{A},\boldsymbol{b}}(x) = \boldsymbol{A}.\boldsymbol{x}\oplus\boldsymbol{b}$. This family was shown to be 2-universal in [2]. For a hash function $h \in H_{XOR}(\mathsf{q},\mathsf{p})$, we have that $h(\boldsymbol{x}) = \boldsymbol{y}$ is a system of linear equations modulo 2: $\boldsymbol{A}.\boldsymbol{x}\oplus\boldsymbol{b} = \boldsymbol{y}$. From another perspective, it can be viewed as a boolean formula $\psi = \bigwedge_{i=1}^{\mathsf{p}}(\bigoplus_{j=1}^{\mathsf{q}}(\boldsymbol{A}[i,j]\wedge\boldsymbol{x}[j]))\oplus\boldsymbol{b}[i] = \boldsymbol{y}[i]$. The solutions to this formula are exactly the elements of the set $h^{-1}(\boldsymbol{y})$.

Gaussian Elimination

Solving a system of linear equations over q variables and p constraints can be done by row reduction technique known variously as *Gaussian Elimination* or *Gauss-Jordan Elimination*. A matrix is in *Row-Echelon form* if rows with at least one nonzero element are above any rows of all zeros. The matrix is in *Reduced Row-Echelon form* if, in addition, every leading non-zero element in a row is 1 and is the only nonzero entry in its column. We refer to the

technique for obtaining the *Reduced Row-Echelon* form of a matrix as Gaussian Elimination. We refer the reader to any standard text on linear algebra (cf., [25]) for details. For a matrix in Reduced Row-Echelon form, the row-rank is simply the number of non-zero rows.

For a system of linear equations $\boldsymbol{A}.\boldsymbol{x} \oplus \boldsymbol{b} = \boldsymbol{y}$, if the row-rank of the augmented matrix is same as row-rank of $\boldsymbol{A}$, then the system is consistent and the number of solutions is $2^{\mathsf{q}-rowrank(\boldsymbol{A})}$ where q is the number of variables in the system of equations. Moreover, if $\boldsymbol{A}$ is in Reduced Row-Echelon form, then the values of the variables corresponding to leading 1s in each row are completely determined by the values assigned to the remaining variables. The variables corresponding to the leading 1's are called *dependent* variables and the remaining variables are *free*. Let X_F and $X \setminus X_F$ denote the set of free and dependent variables respectively. Let $f = |X_F|$. Clearly $f = \mathsf{q} - rowrank(\boldsymbol{A})$. For each possible assignment to the free variables we get an assignment to the dependent variables by propagating the values through the augmented matrix in $\mathcal{O}(\mathsf{q}^2)$ time. Thus we can enumerate all 2^f satisfying assignments to a system of linear equations $\boldsymbol{A}.\boldsymbol{x} \oplus \boldsymbol{b} = \boldsymbol{y}$ if $\boldsymbol{A}$ in Reduced Row-Echelon form.

Gray Codes

A Gray code [14] is an ordering of 2^l binary numbers for some $l \geq 1$ with the property that every pair of consecutive numbers in the sequence differ in exactly one bit. Thus starting from $\mathbf{0}^l$ we can iteratively construct the entire Gray code sequence by flipping one bit in each step. We assume access to a procedure *nextGrayBit* that in each call returns the position of the next bit that is to be flipped. Such a procedure can be implemented in constant time by a trivial modification of Algorithm L in [19].

3 Related Work

Propositional model counting has been of theoretical as well as practical interest over the years [16, 17, 22, 26]. Early investigations showed that both #CNF and #DNF are #P-complete [28]. Consequently, approximation algorithms have been explored for both problems. A major breakthrough for approximate #DNF was achieved by the seminal work of Karp and Luby [17], which provided a Monte Carlo-based FPRAS for #DNF. The proposed FPRAS was improved by follow-up work of Karp, Luby and Madras [18] and Dagum et al. [5], achieving the best known complexity of $\mathcal{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$. In this work, we bring certain ideas of Karp et al. into the hashing framework with significant adaptations.

For #CNF, early work on approximate counting resulted in hashing-based schemes that required polynomially many calls to an NP-oracle [24, 27]. No practical algorithms materialized from the these schemes due to the impracticality of the underlying NP queries. Subsequent attempts to circumvent hardness led to the development of several hashing and sampling-based approaches that achieved scalability but provided very weak or no guarantees [13, 11]. Due to recent breakthroughs in the design of hashing-based techniques, several tools have been developed recently that can handle formulas involving hundreds of thousands of variables while providing rigorous formal guarantees. Overall, these tools can be broadly classified by their underlying hashing-based technique as: (i) obtain a constant factor approximation and then use identical copies of the input formula to obtain ε approximations [10], or (ii) directly obtain ε guarantees[3, 4]. The first technique when applied to DNF formulas is not an FPRAS. In contrast, Chakraborty, Meel and Vardi [4] recently showed that tools based on the latter approach, such as ApproxMC2, do provide FPRAS for #DNF counting. Chakraborty et al. did not analyze the complexity of the algorithm in their work. We now provide a precise complexity analysis of ApproxMC2 for

#DNF. To that end, we first describe the ApproxMC framework on which ApproxMC2 is built.

3.1 ApproxMC Framework

Chakraborty et al. introduced in [3] a hashing-based framework called ApproxMC that requires linear (in n) number of SAT calls. Subsequently in ApproxMC2, the number of SAT calls was reduced from linear to logarithmic (in n). The core idea of ApproxMC is to employ 2−universal hash functions to partition the solution space into *roughly equal small* cells, wherein a cell is called *small* if it has less than or equal to hiThresh solutions, such that hiThresh is a function of ε. A SAT solver is employed to check if a cell is small by enumerating solutions one-by-one until either there are no more solutions or we have already enumerated $\mathsf{hiThresh}+1$ solutions. Following the terminology of [3], we refer to the above described procedure as BSAT (bounded SAT). To determine the number of cells, ApproxMC performs a search that requires $\mathcal{O}(\log \mathsf{n})$ steps and the estimate is returned as the count of the solutions in a randomly picked small cell scaled by the total number of cells. To amplify confidence to the desired levels of $1-\delta$, ApproxMC invokes the estimation routine $\mathcal{O}(\log \frac{1}{\delta})$ times and reports the median of all such estimates. Hence, the number of BSAT invocations is $\mathcal{O}(\log \mathsf{n} \log(\frac{1}{\delta}))$.

FPRAS for #DNF

The key insight of Chakraborty et al. [4] is that the BSAT procedure can be done in polynomial time when the input formula to ApproxMC is in DNF. In particular, the input to every invocation of BSAT is a formula that is a conjunction of the input DNF formula and a set of XOR constraints derived from the hash function. Chakraborty et al. observed that one can iterate over all the cubes of the input formula, substitute each cube into the set of XOR constraints separately, and employ Gaussian Elimination to enumerate the solutions of the simplified XOR constraints. Note that at no step would one have to enumerate more than hiThresh solutions. Since Gaussian Elimination is a polynomial-time procedure, BSAT can be accomplished in polynomial time as well. Chakraborty et al. did not provide a precise complexity analysis of BSAT. We now provide such an analysis. For lack of space we defer all proofs to the full version. The following lemma states the time complexity of the BSAT routine.

▶ **Lemma 1.** *The complexity of* BSAT *when the input formula to* ApproxMC2 *is in DNF is* $\mathcal{O}(\mathsf{mn}^3 + \mathsf{mn}^2/\varepsilon^2)$.

We can now complete the complexity analysis:

▶ **Lemma 2.** *The complexity of* ApproxMC2 *is* $\mathcal{O}((\mathsf{mn}^3 + \mathsf{mn}^2/\varepsilon^2) \log \mathsf{n} \log(1/\delta))$ *when the input formula is in DNF.*

4 Efficient Hashing-based DNF Counter

We now present three key novel algorithmic innovations that allow us to design hashing-based FPRAS for #DNF with complexity similar to Monte Carlo-based state-of-the-art techniques. We first introduce a new family of 2-universal hash functions that allow us to circumvent the need for expensive Gaussian Elimination. We then discuss the concept of Symbolic Hashing, which allows us to design hash functions over a space different than the assignment space, allowing us to achieve significant reduction in the complexity of search procedure for the

Algorithm 1 enumNextREX$(\mathbf{D}, \boldsymbol{u}, \boldsymbol{v}, k)$

1: $\boldsymbol{v}' \leftarrow \boldsymbol{v}$;
2: $\boldsymbol{v}'[k] \leftarrow \neg \boldsymbol{v}[k]$;
3: $\boldsymbol{u}' \leftarrow \boldsymbol{u} \oplus \mathbf{D}[., k]$;
4: **return** $(\boldsymbol{u}', \boldsymbol{v}')$

number of the cells. Finally, we show that BSAT can be replaced by an efficient stochastic estimator. These three techniques allow us to achieve significant reduction in the complexity of hashing-based DNF counter without loss of theoretical guarantees.

4.1 Row-Echelon XOR Hash Functions

The complexity analysis presented in Section 3 shows that the expensive Gaussian Elimination contributes significantly to poor time complexity of ApproxMC2. Since the need for Gaussian Elimination originates from the usage of H_{XOR}, we seek a family of 2-universal hash functions that circumvents this need. We now introduce a Row-Echelon XOR family of hash functions defined as $H_{REX}(\mathsf{q}, \mathsf{p}) = \{\boldsymbol{A}.\boldsymbol{x} \oplus \boldsymbol{b} \mid \boldsymbol{A}^{[\mathsf{p}\times\mathsf{q}]} = [\boldsymbol{I}^{[\mathsf{p}\times\mathsf{p}]} : \mathbf{D}^{[\mathsf{p}\times(\mathsf{q}-\mathsf{p})]}]\}$ where $\boldsymbol{I}$ is the identity matrix, $\mathbf{D}$ and $\boldsymbol{b}$ are random $0/1$ matrix and vector respectively. In particular, we ensure that for every $\mathbf{D}[i,j]$ and $\boldsymbol{b}[i]$ we have $\Pr[\mathbf{D}[i,j] = 1] = \Pr[\mathbf{D}[i,j] = 0] = 0.5$ and also $\Pr[\boldsymbol{b}[i] = 1] = \Pr[\boldsymbol{b}[i] = 0] = 0.5$. Note that $\mathbf{D}$ and $\boldsymbol{b}$ completely define a hash function from H_{REX}. The following theorem establishes the desired properties of universality for H_{REX}. The proof is deferred to full version for lack of space.

▶ **Theorem 3.** *H_{REX} is 2-universal.*

The naive way of enumerating satisfying assignments for a given $\mathbf{D}^{[\mathsf{p}\times(\mathsf{q}-\mathsf{p})]}$, $\boldsymbol{b}^{[\mathsf{p}]}$, and $\boldsymbol{y}^{[\mathsf{p}]}$ is to iterate over all 2^f assignments to the free variables in sequence starting from $\mathbf{0}^{[f]}$ to $\mathbf{1}^{[f]}$, where $f = (\mathsf{q} - \mathsf{p})$. For each assignment $\boldsymbol{v}^{[f]}$ to the free variables, the corresponding assignment to the dependent variables $\boldsymbol{u}^{[\mathsf{q}-f]}$ can be calculated as $\boldsymbol{u} = (\mathbf{D}.\boldsymbol{v}) \oplus \boldsymbol{b} \oplus \boldsymbol{y}$, which requires $O(\mathsf{pq})$ time. Can we do better?

We answer the above question positively by iterating over the 2^f assignments to the free variables out of sequence. In particular, we iterate using the Gray code sequence for f bits. The procedure is outlined in enumNextREX (Algorithm 1). The algorithm takes the hash matrix $\mathbf{D}$, an assignment to the free variables $\boldsymbol{v}$, and an assignment to the dependent variables $\boldsymbol{u}$ as inputs, and outputs the next free-variable assignment $\boldsymbol{v}'$ in the Gray sequence and the corresponding assignment $\boldsymbol{u}'$ to the dependent variables. k represents the position of the bit that is changed between $\boldsymbol{v}$ and $\boldsymbol{v}'$. Thus enumNextREX constructs a satisfying assignment to a Row-Echelon XOR hash function in each invocation in $\mathcal{O}(\mathsf{q})$ time.

4.2 Symbolic Hashing

For DNF formulas, R_ϕ can be exponentially sparse compared to U, which is undesirable[2]. It is possible, however, to transform U to another space U' and the solution space R_ϕ to R'_ϕ such that the ratio $|U'|/|R'_\phi|$ is polynomially bounded and $|R_\phi| = |R'_\phi|$. For DNF formulas, the new universe U' is defined as $U' = \{(\boldsymbol{x}, \phi^{Ci}) \mid \boldsymbol{x} \models \phi^{Ci}\}$. Thus, corresponding to each $\boldsymbol{x} \models \phi$ that satisfies cubes $\phi^{Ci_1}, ..\phi^{Ci_{\boldsymbol{x}}}$ in ϕ, we have the states $\{(\boldsymbol{x}, \phi^{Ci_1}), (\boldsymbol{x}, \phi^{Ci_2})..(\boldsymbol{x}, \phi^{Ci_{\boldsymbol{x}}})\}$ in

[2] Number of steps of ApproxMC2 search procedure increases with sparsity

Algorithm 2 SymbolicDNFApproxMCCore(ϕ, hiThresh)

1: $\mathsf{w} \leftarrow$ width of cubes;
2: $\mathsf{q} \leftarrow \mathsf{n} - \mathsf{w} + \log \mathsf{m}$;
3: $sI \leftarrow \mathsf{n} - \mathsf{w} - \log \mathsf{hiThresh}$;
4: $(\hat{\mathbf{D}}^{[(\mathsf{q}-1)\times(\mathsf{q}-sI)]}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}) \leftarrow \mathsf{SampleBase}(\mathsf{q}, sI)$;
5: $\mathsf{p} \leftarrow \mathsf{LogSATSearch}(\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}, \mathsf{hiThresh}, sI, \mathsf{q}-1)$;
6: $\mathsf{solCount} \leftarrow \mathsf{BSAT}(\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}, \mathsf{hiThresh}, \mathsf{p}, \mathsf{q}, sI)$;
7: **return** $(2^{\mathsf{p}}, \mathsf{solCount})$

U'. Next, the solution space is defined as $R'_\phi = \{(\boldsymbol{x}, \phi^{Ci}) \mid \boldsymbol{x} \models \phi^{Ci} \text{ and } \forall j < i, \boldsymbol{x} \not\models \phi^{Cj}\}$ for a fixed ordering of the cubes. The definition of R'_ϕ ensures that $|R_\phi| = |R'_\phi|$. This transformation is due to Karp and Luby [17].

The key idea of Symbolic Hashing is to perform 2-universal hashing symbolically over the transformed space. In particular, the sampled hash function partitions the space U' instead of U. Therefore, we employ hash functions from $H_{REX}(\mathsf{q}, \mathsf{p})$ over $\mathsf{q} = \mathsf{n} - \mathsf{w} + \log \mathsf{m}$ variables instead of n variables. Note that the variables of a satisfying assignment $\boldsymbol{z} \in \{0,1\}^{\mathsf{q}}$ to the hash function are now different from the variables to a satisfying assignment $\boldsymbol{x} \in \{0,1\}^{\mathsf{n}}$ of the input formula ϕ. We interpret $\boldsymbol{z}$ as follows: the last $\log \mathsf{m}$ bits of $\boldsymbol{z}$ are converted to a number i such that $1 \leq i \leq \mathsf{m}$. Now ϕ^{Ci} corresponds to a partial assignment of $width[\phi^{Ci}]$ variables in that cube. For simplicity, we assume that each cube is of the same width w.[3] The remaining $\mathsf{n} - \mathsf{w}$ bits of $\boldsymbol{z}$ are interpreted to be the assignment to the $\mathsf{n} - \mathsf{w}$ variables not in ϕ^{Ci} giving a complete assignment $\boldsymbol{x}$. Thus we get a pair $(\boldsymbol{x}, \phi^{Ci})$ from $\boldsymbol{z}$ such that $\boldsymbol{x} \models \phi^{Ci}$. For a fixed ordering of variables and cubes we see that there is a bijection between $(\boldsymbol{x}, \phi^{Ci})$ and $\boldsymbol{z}$ and hence the 2-universality guarantee holds over the partitioned space of U'.

4.3 Stochastic Cell-Counting

To estimate the number of solutions in a cell, we need to check for every tuple $(\boldsymbol{x}, \phi^{Ci})$ generated using symbolic hash function as described above: if $(\boldsymbol{x}, \phi^{Ci}) \in R'_\phi$. Such a check would require iteration over cubes ϕ^{Cj} for $1 \leq j \leq (i-1)$ and returning no if $\boldsymbol{x} \models \phi^{Cj}$ for some j and yes otherwise. This would result in procedure with $O(\mathsf{mn})$ complexity.

Our key observation is that a precise count of the number of solutions in a cell is not required and therefore, one can employ a stochastic estimator for the number of solutions in a cell. We proceed as follows: we define the *coverage* of an assignment $\boldsymbol{x}$ as $cov(\boldsymbol{x}) = \{j | \boldsymbol{x} \models \phi^{Cj}\}$. Note that $\sum_{(\boldsymbol{x}, \phi^{Ci}) \in U'} \frac{1}{|cov(\boldsymbol{x})|} = |R_\phi|$.

We define a random variable $\mathsf{c}_{\boldsymbol{x}}$ as the number of steps taken to uniformly and independently sample from $\{1, 2, \ldots, \mathsf{m}\}$, a number j such that $\boldsymbol{x} \models \phi^{Cj}$. For a randomly chosen j, the probability $\Pr[\boldsymbol{x} \models \phi^{Cj}] = |cov(\boldsymbol{x})|/\mathsf{m}$, which follows the Bernoulli distribution. The random variable $\mathsf{c}_{\boldsymbol{x}}$ is the number of Bernoulli trials for the first success, which follows the geometric distribution. Therefore, $\mathsf{E}[\mathsf{c}_{\boldsymbol{x}}] = \mathsf{m}/|cov(\boldsymbol{x})|$, and $\mathsf{E}[\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}] = 1/|cov(\boldsymbol{x})|$. The estimator $\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$ has been previously employed by Karp et al. [18]. Here, we show that it can also be used for Stochastic Cell-Counting: we define the estimator for the number of solutions in a cell as $\Omega_{\boldsymbol{y}} = \sum_{(\boldsymbol{x}, \phi^{Ci}) \in h^{-1}(\boldsymbol{y})} \mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$.

[3] We can handle non-uniform width cubes by sampling ϕ^{Ci} with probability $\frac{2^{\mathsf{n}-width[\phi^{Ci}]}}{\sum_{j=1}^{\mathsf{m}} 2^{\mathsf{n}-width[\phi^{Cj}]}}$ instead of uniformly

Algorithm 3 SymbolicDNFApproxMC$(\phi, \varepsilon, \delta)$

1: hiThresh $\leftarrow 2 * (1 + 9.84\left(1 + \frac{\varepsilon}{1+\varepsilon}\right)\left(1 + \frac{1}{\varepsilon}\right)^2)$;
2: $t \leftarrow \lceil 17 \log_2(3/\delta) \rceil$;
3: EstimateList $\leftarrow$ emptyList; iter $\leftarrow 0$;
4: **repeat**
5: iter $\leftarrow$ iter $+ 1$;
6: (cellCount, solCount) $\leftarrow$ SymbolicDNFApproxMCCore$(\phi,$ hiThresh$)$;
7: **if** (cellCount $\neq \perp$) **then** AddToList(EstimateList, solCount $\times$ cellCount);
8: **until** (iter $\geq t$);
9: finalEstimate $\leftarrow$ FindMedian(EstimateList);
10: **return** finalEstimate

Algorithm 4 SampleBase(q, sI)

1: Sample $\mathbf{G}$ uniformly from $\{0,1\}^{[sI \times (\mathsf{q}-sI)]}$;
2: Sample uniformly an upper triangular matrix $\mathbf{E}^{[(\mathsf{q}-sI-1)\times(\mathsf{q}-sI)]}$ with $\mathbf{E}[i,i] = 1$ for all i.
3: $\hat{\mathbf{D}} \leftarrow [\frac{\mathbf{G}}{\mathbf{E}}]$;
4: Sample $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ uniformly from $\{0,1\}^{\mathsf{q}-1}$;
5: **return** $\hat{\mathbf{D}}, \hat{b}, \hat{y}$

4.4 The Full Algorithm

We now incorporate the above techniques into ApproxMC2 and call the revised algorithm SymbolicDNFApproxMC, which is presented as Algorithm 3. First, note that expression for hiThresh is twice that for ApproxMC2. Then, in line 4, a matrix $\hat{\mathbf{D}}$ and vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ are obtained, which are employed to construct an appropriate hash function and cell during the search procedure of SymbolicDNFApproxMCCore. SymbolicDNFApproxMC makes $t = O(\log(1/\delta))$ calls to SymbolicDNFApproxMCCore (line 4-8) and returns median of all the estimates (lines 9-10) to boost the probability of success to $1 - \delta$.

We now discuss the subroutine SymbolicDNFApproxMCCore, which is an adaptation of ApproxMC2Core but with significant differences. First, for DNF formulas with cube width w, the number of solutions is lower bounded by $2^{\mathsf{n}-w}$. Therefore, instead of starting with 1 hash constraint, we can safely start with $sI = \mathsf{n} - w - \log \mathsf{hiThresh}$ constraints (lines 3-4). Thereafter, SymbolicDNFApproxMCCore calls LogSATSearch in line 5 to find the right number p of constraints. The cell count with p constraints is calculated in line 6 and the estimate $(2^{\mathsf{p}}, \mathsf{solCount})$ is returned in line 7.

SampleBase algorithm constructs the base matrix $\hat{\boldsymbol{D}}$ and base vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ required for sampling from H_{REX} family. $\mathbf{G}$ is a random matrix of dimension $sI \times (\mathsf{q} - sI)$ and $\mathbf{E}$ is a random upper triangular matrix of dimension $(\mathsf{q} - sI - 1) \times (\mathsf{q} - sI)$ with all diagonal elements 1. In line 3, $\hat{\boldsymbol{D}}$ is constructed as the vertical concatenation $[\frac{\mathbf{G}}{\mathbf{E}}]$.

LogSATSearch (algorithm 5) performs a binary search to find the number of constraints p at which the cell count falls below hiThresh. For DNF formula with cube width of w, since the number of solutions is bounded between $2^{\mathsf{n}-w}$ and $\mathsf{m} * 2^{\mathsf{n}-w}$, we need to perform search for p between $\mathsf{n} - w$ and $\mathsf{n} - w + \log \mathsf{m}$. Therefore, binary search can take at most $O(\log \log \mathsf{m})$ steps to find correct p.

Symbolic Hashing is implemented in Algorithm 6 (BSAT). In line 2, we obtain a hash function from $H_{REX}(\mathsf{q}, \mathsf{p})$ over $\mathsf{q} = \mathsf{n} - \mathsf{w} + \log \mathsf{m}$ variables by calling Extract. We assume access to a procedure *nextGrayBit* in line 10 that returns the position of the bit that is

Algorithm 5 LogSATSearch($\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}$, hiThresh, low, hi)

```
1: lowerFib ← 0; upperFib ← hi − low + 1; p ← low;
2: FailRecord[0] ← 1; FailRecord[hi − low + 1] ← 0;
3: FailRecord[i] ← ⊥ for all i other than 0 and hi − low + 1;
4: while true do
5:     C_BSAT ← BSAT(φ, D̂, b̂, ŷ, hiThresh, p, q, sI);
6:     if (C_BSAT ≥ hiThresh) then
7:         if (FailRecord[p + 1 − low + 1] = 0) then return p + 1;
8:         FailRecord[i] ← 1 for all i ∈ {1, ... p − low + 1};
9:         lowerFib ← p − low + 1;
10:        p ← (upperFib + lowerFib)/2;
11:    else
12:        if (FailRecord[p − 1 − low + 1] = 1) then return p;
13:        FailRecord[i] ← 0 for all i ∈ {p, ... hi − low + 1};
14:        upperFib ← p − low + 1;
15:        p ← (upperFib + lowerFib)/2;
```

Algorithm 6 BSAT($\phi, \hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}$, hiThresh, p, q, sI)

```
1: count ← 0;
2: D, b, y ← Extract(D̂, b̂, ŷ, p, q, sI);
3: u^p ← b ⊕ y;
4: v^(q−p) ← 0^(q−p);
5: for (j = 0; j < 2^(q−p); j++) do
6:     z ← [u : v];
7:     (x, φ^Ci) = interpret(z);
8:     count = count + CheckSAT(x, φ^Ci, count, hiThresh);
9:     if count ≥ hiThresh then return hiThresh;
10:    k ← nextGrayBit(q − p, j);
11:    (u, v) ← enumNextREX(D, u, v, k);
12: return count
```

flipped between two consecutive assignments. A satisfying assignment $\boldsymbol{z}$ to the hash function is constructed in line 6. $\boldsymbol{z}$ is interpreted to generate a pair $(\boldsymbol{x}, \phi^{Ci})$ in line 7 which is checked for satisfiability in line 8. The final cell count is returned in line 12.

In CheckSAT (algorithm 7), we implement the stochastic cell counting procedure. The key idea is to sample cubes uniformly at random from $\{1, 2 \ldots \mathsf{m}\}$ till a cube ϕ^{Cj} is found such that $\boldsymbol{x} \models \phi^{Cj}$ (lines 2-5). The number of cubes sampled $\mathsf{c}_{\boldsymbol{x}}$ divided by total number of cubes m is the estimate returned (line 6).

Procedure LogSATSearch in SymbolicDNFApproxMC is based upon the LogSATSearch in ApproxMC2[4]. As noted in the analysis of ApproxMC2, such a logarithmic search procedure requires that the solution space for a hash function with $\mathsf{p}+1$ hash constraints is a subset of the solution space with p hash constraints. Furthermore, we want to preserve Row-Echelon nature of the resulting hash constraints. To this end, we first construct $\mathbf{D}^{[\mathsf{q}\times(\mathsf{q}-\mathsf{p})]}$ and $\boldsymbol{b}^{[\mathsf{q}-1]}$ as follows:

To seed the construction procedure, in SampleBase (algorithm 4) we first randomly sample a 0/1 vector $\hat{\boldsymbol{b}}$ of size $\mathsf{q}-1$ which is the maximum number of hash constraints possible. We

Algorithm 7 CheckSAT($\boldsymbol{x}, \phi^{Ci}, count, \mathsf{hiThresh}$)

1: $\mathsf{c}_{\boldsymbol{x}} \leftarrow 0$;
2: **while** $count + \mathsf{c}_{\boldsymbol{x}}/m < \mathsf{hiThresh}$ **do**
3: Uniformly sample j from $\{1, 2, .., \mathsf{m}\}$;
4: $\mathsf{c}_{\boldsymbol{x}} \leftarrow \mathsf{c}_{\boldsymbol{x}} + 1$;
5: **if** $\boldsymbol{x} \models \phi^{Cj}$ **then**
6: **return** $\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$;
7: **return** $\mathsf{c}_{\boldsymbol{x}}/\mathsf{m}$

Algorithm 8 Extract($\hat{\mathbf{D}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{y}}, \mathsf{p}, \mathsf{q}, sI$)

1: $\hat{\mathbf{D}}' \leftarrow \hat{\mathbf{D}}[0 : \mathsf{p}, 0 : \mathsf{q} - sI]$; $\boldsymbol{b} \leftarrow \hat{\boldsymbol{b}}[0 : \mathsf{p}]$;
2: $\boldsymbol{y} \leftarrow \hat{\boldsymbol{y}}[0 : \mathsf{p}]$;
3: **for** $(i = sI;\ i < \mathsf{p}; i++)$ **do**
4: **for** $(j = 0;\ j < i; j++)$ **do**
5: **if** $\hat{\mathbf{D}}'[j, (i - sI)] == 1$ **then**
6: $\hat{\mathbf{D}}'[j, .] \leftarrow \hat{\mathbf{D}}'[j, .] \oplus \hat{\mathbf{D}}'[i, .]$;
7: $\boldsymbol{b}[j] \leftarrow \boldsymbol{b}[j] \oplus \boldsymbol{b}[i]$;
8: $\boldsymbol{y}[j] \leftarrow \boldsymbol{y}[j] \oplus \boldsymbol{y}[i]$;
9: $\mathbf{D} \leftarrow \hat{\mathbf{D}}'[0 : \mathsf{p}, \mathsf{p} - sI : \mathsf{q} - sI]$;
10: **return** $\mathbf{D}, \boldsymbol{b}, \boldsymbol{y}$

then construct a 0/1 matrix $\hat{\mathbf{D}}$ as follows: $\hat{\mathbf{D}}^{[(\mathsf{q}-1)\times(\mathsf{q}-sI)]} = [\frac{\mathbf{G}}{\mathbf{E}}]$ where matrix $\mathbf{G}^{[sI\times(\mathsf{q}-sI)]}$ is a random 0/1 matrix with sI rows, and matrix $\mathbf{E}^{[(\mathsf{q}-sI)\times(\mathsf{q}-sI)]}$ is defined as as follows:

- $\mathbf{E}[i, j] = 1$ if $i = j$
- $\mathbf{E}[i, j] = 0$ if $i > j$
- $\Pr[\mathbf{E}[i, j] = 1] = \Pr[\mathbf{E}[i, j] = 0] = 0.5$ if $i < j$

The reason for this definition of $\hat{\mathbf{D}}$ is that for DNF counting we have a good lower bound on the number of hash constraints we can start with. The number of rows in $\mathbf{G}$ corresponds to this lower bound. The definition of $\mathbf{E}$ ensures that the rows of $\mathbf{E}$ are linearly independent which results in a monotonically shrinking solution space.

The Extract procedure (algorithm 8) takes $\hat{\mathbf{D}}$,$\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ and a number p as input and returns $\mathbf{D}$,$\boldsymbol{b}$ and cell $\boldsymbol{y}$ such that $(\mathbf{D}, \boldsymbol{b})$ represents a hash function from H_{REX} with p constraints and $\boldsymbol{y}$ represents a cell. A precondition for Extract is $sI \leq \mathsf{p} \leq \mathsf{q} - 1$. In lines 1 and 2, the first p rows of $\hat{\mathbf{D}}$ and first p elements of $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{y}}$ are selected as $\hat{\mathbf{D}}'$, $\boldsymbol{b}$ and $\boldsymbol{y}$ respectively. The first sI rows of $\hat{\mathbf{D}}'$ form the matrix $\mathbf{G}$ in the definition of $\hat{\mathbf{D}}$ and the remaining $\mathsf{p} - sI$ rows of $\hat{\mathbf{D}}'$ are the first $\mathsf{p} - sI$ rows of matrix $\mathbf{E}$. Each row from sI to p is used to reduce the preceding rows in lines 5 to 8 so that the only non-zero elements of the first $\mathsf{p} - sI$ columns are the leading 1s in rows sI to p. Thus Extract ensures that for a given $\hat{\mathbf{D}}$,$\boldsymbol{b}$ and $\hat{\boldsymbol{y}}$, the solution space of $\mathbf{D}^{[\mathsf{p}\times(\mathsf{q}-\mathsf{p})]}$,$\boldsymbol{b}^{[\mathsf{p}]}$ and $\boldsymbol{y}^{[\mathsf{p}]}$ is a superset of solution space of $\mathbf{D}^{[(\mathsf{p}+1)\times(\mathsf{q}-\mathsf{p}-1)]}$,$\boldsymbol{b}^{[\mathsf{p}+1]}$ and $\boldsymbol{y}^{[\mathsf{p}+1]}$ for all p.

5 Analysis

In order to prove the correctness of SymbolicDNFApproxMC, we first state the following helper lemma. We defer the proofs to the full version due to lack of space.

▶ **Lemma 4.** *For every* $1 \leq \mathsf{p} \leq \mathsf{q}$ *and let* $\mu_{\mathsf{p}} = |R_\phi|/2^{\mathsf{p}}$. *For every* $\beta > 0$ *and* $0 < \varepsilon < 1$ *we have*

1. $\Pr[|\Omega_{\boldsymbol{y}} - \mu_{\mathsf{p}}| > \frac{\varepsilon}{(1+\varepsilon)}\mu_{\mathsf{p}}] \leq \frac{2}{\frac{\varepsilon^2}{(1+\varepsilon)^2}\mu_{\mathsf{p}}}$
2. $\Pr[\Omega_{\boldsymbol{y}} \leq \beta\mu_{\mathsf{p}}] \leq \frac{2}{2+(1-\beta^2)\mu_{\mathsf{p}}}$

The difference in lemma 4 and lemma 1 in [4] is that the probability bounds differ by a factor of 2. We account for this difference by making hiThresh in SymbolicDNFApproxMC twice the value of hiThresh in ApproxMC2. Therefore the rest of the proof of Theorem 7 (below) is exactly the same as the proof of Theorem 4 of [4]. For completeness, we first restate lemmas 2 and 3 from [4] below.

In the following, T_{p} denotes the event $(\Omega_{\boldsymbol{y}} < \mathsf{hiThresh})$, and L_{p} and U_{p} denote the events $(\Omega_{\boldsymbol{y}} < \frac{|R_\phi|}{(1+\varepsilon)2^{\mathsf{p}}})$ and $(\Omega_{\boldsymbol{y}} > \frac{|R_\phi|}{2^{\mathsf{p}}}(1 + \frac{\varepsilon}{1+\varepsilon}))$ respectively. p^* denotes the integer $\lfloor \log_2 |R_\phi| - \log_2(4.92(1 + \frac{1}{\varepsilon})^2) \rfloor$

▶ **Lemma 5.** *The following bounds hold:* 1) $\Pr[T_{\mathsf{p}^*-3}] \leq \frac{1}{62.5}$ 2) $\Pr[L_{\mathsf{p}^*-2}] \leq \frac{1}{20.68}$ 3) $\Pr[L_{\mathsf{p}^*-1}] \leq \frac{1}{10.84}$ 4) $\Pr[L_{\mathsf{p}^*} \cup U_{\mathsf{p}^*}] \leq \frac{1}{4.92}$

Let B denote the event that SymbolicDNFApproxMC returns a pair $(2^{\mathsf{p}}, nSols)$ such that $2^{\mathsf{p}} * nSols$ does not lie in the interval $[\frac{|R_\phi|}{1+\varepsilon}, |R_\phi|(1+\varepsilon)]$.

▶ **Lemma 6.** $\Pr[B] \leq 0.36$

▶ **Theorem 7.** *Let* SymbolicDNFApproxMC$(\phi, \varepsilon, \delta)$ *return count* c. *Then* $\Pr[|R_\phi|/(1+\varepsilon) \leq c \leq (1+\varepsilon)|R_\phi|] \geq 1 - \delta$.

Theorem 7 follows from lemmas 4, 5 and 6 and noting that SymbolicDNFApproxMC boosts the probability of correctness of the count returned by SymbolicDNFApproxMCCore to $1 - \delta$ by using median of $t = O(\log(1/\delta))$ calls.

▶ **Theorem 8.** SymbolicDNFApproxMC *runs in* $\tilde{O}(\mathsf{mn}\log(1/\delta)/\varepsilon^2)$ *time.*[4]

6 Conclusion

Hashing-based techniques have emerged as a promising approach to obtain counting algorithms and tools that scale to large instances while providing strong theoretical guarantees. This has led to an interest in designing hashing-based algorithms for counting problems that are known to be amenable to fully polynomial randomized approximation schemes. The prior hashing-based approach [4] provided FPRAS for DNF but with complexity much worse than state-of-the-art techniques. In this work, we introduced (i) Symbolic Hashing, (ii) Stochastic Cell-Counting, and (iii) a new 2-universal family of hash functions, and obtained a hashing-based FPRAS for #DNF with complexity similar to state-of-the-art.

Given the recent interest in hashing-based techniques and generality of our contributions, we believe concepts introduced in this paper can lead to design of hashing-based techniques for other classes of constraints. For example, all prior versions of ApproxMC relied on deterministic SAT solvers for exactly counting the solutions in a cell for #CNF. The technique of Stochastic Cell-Counting opens up the door for the usage of probabilistic SAT solvers for #CNF. Furthermore, a salient feature of the H_{REX} family is the sparsity of its hash functions. In fact, the sparsity increases with the addition of constraints. Sparse hash functions have been shown to be desirable for efficiently solving CNF+XOR constraints [15, 9, 12]. An interesting direction for future work is to test H_{REX} family with CNF formulas.

[4] We say $f(n) \in \tilde{O}(g(n))$ if $\exists k : f(n) \in \mathcal{O}(g(n)\log^k(g(n))$

Acknowledgements. The authors thank Jeffrey Dudek, Supratik Chakraborty and Dror Fried for valuable discussions.

References

1 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 340–351. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238208.

2 J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

3 S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable approximate model counter. In *Proc. of CP*, pages 200–216, 2013.

4 S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*, 2016.

5 Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An optimal algorithm for monte carlo estimation. *SIAM Journal on computing*, 29(5):1484–1496, 2000.

6 Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 16(4):523–544, 2007.

7 C. Domshlak and J. Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research*, 30(1):565–620, 2007.

8 Leonardo Duenas-Osorio, Kuldeep S Meel, Roger Paredes, and Moshe Y Vardi. Counting-based reliability estimation for power-transmission grids. In *AAAI*, pages 4488–4494, 2017.

9 S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Low-density parity constraints for hashing-based discrete integration. In *Proc. of ICML*, pages 271–279, 2014.

10 Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*, pages 334–342, 2013.

11 V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *Proc. of the AAAI*, volume 22, page 198, 2007.

12 C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. Short XORs for Model Counting; From Theory to Practice. In *SAT*, pages 100–106, 2007.

13 C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, volume 21, pages 54–61, 2006.

14 Frank Gray. Pulse code communication, 17 1953. US Patent 2,632,058.

15 Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, 21(1):41–58, 2016. doi:10.1007/s10601-015-9204-z.

16 M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2-3):169–188, 1986. URL: http://portal.acm.org/citation.cfm?id=11534.11537.

17 R.M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. *Proc. of FOCS*, 1983.

18 R.M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

19 Donald E Knuth. Generating all n-tuples. *The Art of Computer Programming*, 4, 2004.

20 James D Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, pages 101–133, 2006.

21 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996. doi:10.1016/0004-3702(94)00092-1.

22 T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. of SAT*, 2004.

23 T. Sang, P. Beame, and H. Kautz. Performing bayesian inference by weighted model counting. In *Prof. of AAAI*, pages 475–481, 2005.

24 L. Stockmeyer. The complexity of approximate counting. In *Proc. of STOC*, pages 118–126, 1983.

25 Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

26 S. Toda. On the computational power of PP and (+)P. In *Proc. of FOCS*, pages 514–519. IEEE, 1989.

27 L. Trevisan. Lecture notes on computational complexity. *Notes written in Fall*, 2002. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.9877&rep=rep1&type=pdf`.

28 L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

Average Stack Cost of Büchi Pushdown Automata*†

Jakub Michaliszyn[1] and Jan Otop[2]

1 **University of Wrocław, Poland**
2 **University of Wrocław, Poland**

Abstract

We study the average stack cost of Büchi pushdown automata (Büchi PDA). We associate a non-negative price with each stack symbol and define the cost of a stack as the sum of costs of all its elements. We introduce and study the average stack cost problem (ASC), which asks whether there exists an accepting run of a given Büchi PDA such that the long-run average of stack costs is below some given threshold. The ASC problem generalizes mean-payoff objective and can be used to express quantitative properties of pushdown systems. In particular, we can compute the average response time using the ASC problem. We show that the ASC problem can be solved in polynomial time.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases pushdown automata, average stack cost, weighted pushdown systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.42

1 Introduction

Weighted pushdown systems (WPSs) combine finite-control, unbounded stack and weights on transitions. Weights are aggregated using semiring operations [10] or the long-run average [5]. These features make them a powerful formalism capable of expressing interesting program properties [10, 9]. Still, WPSs considered in the literature fall short of expressing the following basic quantitative specification.

Consider the following *client-server scenario*, consisting of two agents, a server and a client. The client sends requests (r), which are granted (g) by the server. Each grant satisfies all pending requests. All other events are abstracted to a null instruction (#). We are interested in checking properties of such systems over infinite runs. We are only interested in sequences with infinitely many requests and grants. The average workload property (AW) for client-server scenario, defined as the long-run average of the number of pending requests over all positions, was studied in [4].

WPSs can model the client-server scenario, but they cannot express AW for two reasons. First, WPSs considered in the literature [5] have no Büchi acceptance condition, and hence we cannot specify traces with infinitely many requests and grants. Second, weights in WPSs are bounded, and hence the long-run average is bounded by the maximal weight, whereas AW is unbounded.

In this paper we study WPSs with Büchi acceptance conditions (known as Büchi pushdown automata) and unbounded weights depending on the stack content, called stack costs. More

* This work was supported by the National Science Centre (NCN), Poland under grant 2014/15/D/ST6/04543.

† A full version of the paper is available at [7], `https://arxiv.org/abs/1710.04490`.

precisely, we define the stack cost as a non-negative linear combination of the number of occurrences of every stack letter, i.e., given stack pricing that assigns a non-negative cost with stack symbols, the stack cost is the sum of prices of its elements. We investigate the *average stack cost* (ASC) during infinite computations of an Büchi pushdown automaton. For a finite computation, the average stack cost is simply the sum of the stack costs in every position divided by the number of positions. It is extended to infinite computations by taking the limit of the average stack costs of all the (finite) prefixes of this infinite computation. As the limit may be undefined (when the sequence of prefixes diverge), we consider two values, the limit inferior and limit superior over all prefixes.

We argue that with the ASC problem we can express interesting system properties. In particular, we can express AW from the client-server scenario. Moreover, we can express a variant of AW where each grant satisfies only one request. This variant of AW cannot be specified with models from [4]. We can also use ASC to compute the average response time property [3], which asks for the average number of steps between a request and the corresponding grant. In this variant of the average response time, we can assume that each grant satisfies one request, which has not been possible in previous formalisms [3].

Contributions. The main results presented in this paper are as follows.

- The average stack cost problem can be solved in polynomial time assuming unary encoding of stack pricing.
- One-player games on WPSs with the conjunctions of mean-payoff and Büchi objectives can be solved in polynomial time, even assuming binary encoding of weights.
- The average response time property over WPSs in a variant of the client-server scenario where each grant satisfies only one request can be computed in polynomial time.

Overview. We start with basic definitions in Section 2. Next, in Section 3 we discuss convergence of the partial averages of the stack costs. In Section 4, we show that to solve ASC we can bound the stack costs along the whole run. This allows us to reduce ASC to the average letter cost problem, which is equivalent to one-player games on WPSs with the conjunction of mean-payoff and Büchi objectives. We chose letter-based formalization rather than WPSs with weights on transitions, as it allows us to use classical language-theoretic results on ω-PDA. We apply these results in Section 5 to show that the average letter cost problem can be solved in polynomial time. Finally, we discuss the connection between ASC and the average response time property (Section 6).

The detailed proofs are presented in the full version of this paper [7].

Related work. WPSs with weights from a *bounded idempotent semiring* and their applications have been studied in [10, 9]. In bounded idempotent semirings there are no infinite descending chains, e.g., the natural numbers, in contrast to the integers. The results from [9] have been generalized to WPSs over indexed domains [8], which still do not capture the integers. WPSs with integer weights aggregated with the long-run average operation (a.k.a. mean-payoff objective) have been studied in [5]. It has been shown that one-player games on WPSs with mean-payoff objective can be solved in polynomial time.

The average stack cost is closely related to the average energy objective studied over finite graphs [1]. In contrast to stack cost, energy levels are not observable, i.e., transitions do not depend on energy levels. One player energy games are decidable in polynomial time. As we can express energy levels using stack costs, the results of this paper can be considered as a generalization of the average-energy objective in the one-player case. However, two-player

energy games are decidable in NP ∩ coNP [1], while even mean-payoff games on WPSs are undecidable [5]. Since the average stack cost generalizes the mean-payoff objectives, two-player average-stack-cost games are undecidable.

2 Preliminaries

Words and automata. Given a finite alphabet Σ of letters, a *word* w is a finite or infinite sequence of letters. We denote the set of all finite words over Σ by Σ^*, and the set of all infinite words over Σ by Σ^ω. We use ϵ to denote the empty word.

For a word w, we define $w[i]$ as the i-th letter of w, and we define $w[i,j]$ as the subword $w[i]w[i+1]\dots w[j]$ of w. We allow $j=\infty$ in $w[i,j]$. By $|w|$ we denote the length of w. We use the same notation for sequences that start from 0.

A *(non-deterministic) pushdown automaton* (PDA) is a tuple $(\Sigma,\Gamma,Q,Q_0,Q_F,\delta)$, where Σ is the input alphabet, Γ is a finite stack alphabet, Q is a finite set of states, $Q_0\subseteq Q$ is a set of initial states, $Q_F\subseteq Q$ is a set of accepting states, and $\delta\subseteq Q\times\Sigma\times(\Gamma\cup\{\bot\})\times Q\times\Gamma^*$ is a finite transition relation. We define Büchi-PDA (called ω-PDA for short) in the same way; these automata differ in semantics. The size of an automaton $\mathcal{A}=(\Sigma,\Gamma,Q,Q_0,Q_F,\delta)$, denoted by $|\mathcal{A}|$, is $|Q|+|\delta|$.

Assume a PDA (resp., ω-PDA) $\mathcal{A}=(\Sigma,\Gamma,Q,Q_0,Q_F,\delta)$. A *configuration* of $\mathcal{A}$ is a tuple $(q,a,u)\in Q\times(\Sigma\cup\{\epsilon\})\times(\Gamma\cup\{\bot\})^*$, where $\bot$ occurs only once in u; it occurs as its first symbol. A *run* π of $\mathcal{A}$ is a sequence of configurations such that $\pi[0]=(q_0,\epsilon,\bot)$ for some $q_0\in Q_0$ and for every $i<|\pi|$, if $\pi[i,i+1]=(q,a,u)(q',a',u')$, we have $\delta(q,a',x,q',y)$ for some x,y such that either $x=u=\bot$ and $u'=y$ or $x\neq\bot$, $u=u_sx$ for some u_s and $u'=u_sy$. Runs of PDA are finite sequences, while runs of ω-automata are infinite.

A run $\pi=(q^0,a^0,u^0)(q^1,a^1,u^1)\dots$ *gives* the word $a^0a^1\dots$. A finite run π of a PDA is *accepting* if the last state in π belongs to Q_F. An infinite run π of an ω-PDA is *accepting* if it visits Q_F infinitely often, i.e., satisfies the Büchi acceptance condition, and gives an infinite word. The *language recognized (or accepted) by the PDA* $\mathcal{A}$ (resp., ω-PDA $\mathcal{A}$), denoted $\mathcal{L}(\mathcal{A})$, is the set of all words given by accepting runs of $\mathcal{A}$.

Weighted pushdown systems. A weighted pushdown system (WPS) $\mathcal{P}$ is pair $(\mathcal{A},\mathrm{wt})$ such that (1) $\mathcal{A}$ is a PDA (resp., ω-PDA) $\mathcal{A}=(\Sigma,\Gamma,Q,Q_0,Q_F,\delta)$, (2) the alphabet Σ is a singleton, (3) all states are accepting, i.e., $Q=Q_F$, and (4) wt is a cost function that maps transitions δ into a cost domain (which is $\mathbb{Z}$ in our case). The alphabet Σ and the set of accepting states are typically omitted.

Context-free grammars (CFG) and their languages. A context-free grammar (CFG) is a tuple $G=(\Sigma,V,S,P)$, where Σ is the alphabet, V is a set of *non-terminals*, $S\in V$ is a *start symbol*, and P is a set of *production rules*. Each production rule p has the following form $v\to u$, where $v\in V$ and $u\in(\Sigma\cup V)^*$. We define *derivation* $\to_G$ as a relation on $(\Sigma\cup V)^*\times(\Sigma\cup V)^*$ as follows: $w\to_G w'$ iff $w=w_1vw_2$, $w'=w_1uw_2$, and $v\to u$ is a production from G. We define $\to_G^*$ as the transitive closure of $\to_G$. The *language generated by* G, denoted by $\mathcal{L}(G)=\{w\in\Sigma^*\mid S\to_G^* w\}$ is the set of words that can be derived from the start symbol S. CFGs and PDAs are language-wise polynomial equivalent (i.e., there is a polynomial time procedure that, given a PDA, outputs a CFG of the same language and vice versa) [6].

2.1 Basic problems

Let $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, Q_F, \delta)$ be an ω-PDA. A *stack pricing* is a function $\mathbf{c} : \Gamma \to \mathbb{N}$ that assigns each stack symbol with a natural number (we assume 0 is natural). We extend $\mathbf{c}$ to configurations (q, a, u) by setting $\mathbf{c}((q, a, u)) = \sum_{i=1}^{|u|} \mathbf{c}(u[i])$, where we assume that $\mathbf{c}(\bot) = 0$.

Given a run π of $\mathcal{A}$, a stack pricing $\mathbf{c}$ and $k > 0$, we define the *average stack cost* of the prefix of π of length k, denoted by $\mathsf{ASC}(\pi, \mathbf{c}, k)$, as $\frac{1}{k} \sum_{i=0}^{k-1} \mathbf{c}(\pi[i])$.

We are interested in establishing the average stack cost for the whole runs, which can be formalized in two ways. The *infimum-average stack cost* of π, denoted by $\mathsf{IASC}(\pi, \mathbf{c})$, and the *supremum-average stack cost* of π, denoted by $\mathsf{SASC}(\pi, \mathbf{c})$, are defined as

$$\mathsf{IASC}(\pi, \mathbf{c}) = \liminf_{k \to \infty} \mathsf{ASC}(\pi, \mathbf{c}, k) \qquad \mathsf{SASC}(\pi, \mathbf{c}) = \limsup_{k \to \infty} \mathsf{ASC}(\pi, \mathbf{c}, k)$$

If $\mathbf{c}$ is known from the contexts, we omit it and write $\mathsf{IASC}(\pi)$ instead of $\mathsf{IASC}(\pi, \mathbf{c})$ and similarly for SASC.

We define two decision questions collectively called the average stack cost problem.

▶ **The `IASC` problem**: given an ω-PDA $\mathcal{A}$, a stack pricing $\mathbf{c}$, $\bowtie \in \{<, \leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists an accepting run π of $\mathcal{A}$ such that $\mathsf{IASC}(\pi, \mathbf{c}) \bowtie \lambda$.

▶ **The `SASC` problem**: given an ω-PDA $\mathcal{A}$, a stack pricing $\mathbf{c}$, $\bowtie \in \{<, \leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists an accepting run π of $\mathcal{A}$ such that $\mathsf{SASC}(\pi, \mathbf{c}) \bowtie \lambda$.

We assume that the numbers in the stack pricing and the threshold are given in unary, i.e., in an instance I of the average stack cost problem, values of $\mathbf{c}$ and λ are polynomially bounded (in the size of the instance).

▶ Remark. Observe that the average stack cost problem generalizes WPSs with mean-payoff objectives. First, WPSs consists of a PDA (resp., ω-PDA) and a cost function from transitions into integers. We can however add a constant C to all weights, which change all mean-payoff values by C. Thus, we can assume that costs are non-negative. Second, we can emulate costs on transitions by extending the stack alphabet with letters corresponding to transitions, and storing the last taken transition at the top of the stack. Hence, allowing, in addition to stack costs, costs on transitions does not change the expressive power or the complexity. For simplicity, we do not consider costs on transitions. Finally, the average stack cost strictly generalizes WPSs with mean-payoff objectives as it can be unbounded whereas the mean-payoff is bounded by the maximal weight of the transition.

▶ **Example 1.** Recall the client-server scenario from the introduction. Assume that the stack alphabet is $\Gamma = \{r\}$ and all requests are pushed on the stack by the client. Then, upon a grant the server empties the stack. Observe that if the cost of a request on the stack is 1, i.e., $\mathbf{c}(r) = 1$, then the average stack cost equals AW.

We can modify this example to model that each grant satisfies a single request. Simply, we require the server to pop only a single request upon a grant. Again, the average stack cost equals AW.

3 Properties of Average Stack Cost

We extend the notion of the average stack cost to PDA, defining $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \inf\{\mathsf{IASC}(\pi, \mathbf{c}) \mid \pi \text{ is an accepting run of } \mathcal{A}\}$ and $\mathsf{SASC}(\mathcal{A}, \mathbf{c}) = \inf\{\mathsf{SASC}(\pi, \mathbf{c}) \mid \pi \text{ is an accepting run of } \mathcal{A}\}$.

We can easily construct a run π and stack pricing $\mathbf{c}$, such that $\mathsf{IASC}(\pi, \mathbf{c}) < \mathsf{SASC}(\pi, \mathbf{c})$. We now show an example proving a stronger claim, stating that even $\mathsf{IASC}(\mathcal{A}, \mathbf{c})$ and $\mathsf{SASC}(\mathcal{A}, \mathbf{c})$ can have different values.

▶ **Example 2.** Consider an automaton $\mathcal{A}$ with three states U, B, A, one alphabet symbol a and two stack symbols α, β, and the stack pricing such that $\mathbf{c}(\alpha) = 0$ and $\mathbf{c}(\beta) = 3$. State A is the only accepting and the only starting state. The transition function is as follows.

$$\delta(A, a, \bot, U, \alpha) \qquad \delta(U, a, \alpha, U, \alpha\alpha) \qquad \delta(U, a, \alpha, B, \beta)$$
$$\delta(B, a, \beta, B, \epsilon) \qquad \delta(B, a, \beta, A, \epsilon) \qquad \delta(B, a, \alpha, B, \beta)$$

Every accepting run of $\mathcal{A}$ starts in the state A, adds some number of symbols α to the stack in state U, and then goes to the state B, where it clears the stack, but to remove a symbol α, it first needs to convert it to (costly) β. Then it reaches A with empty stack and repeats.

Observe that $\mathsf{SASC}(\mathcal{A}, \mathbf{c}) = 1$. To see this, consider an accepting run π. For any position $p > 0$ with an accepting state we have that $\mathsf{ASC}(\pi, \mathbf{c}, p-1) = 1$. To show this, we assign to every β symbol that occur in $\pi[0, p-1]$ three positions: right before it was removed, right before it replaced some α, and right before this α was added. In this way we cover all the positions in $\pi[0, p-1]$, which means that the number of β symbols is three times the number of positions, so $\mathsf{ASC}(\pi, \mathbf{c}, p-1) = 1$.

In contrast, we show that $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = 0$. Let π_i be the sequence of configurations

$$(A, a, \bot)(U, a, \bot\alpha) \dots (U, a, \bot\alpha^i)(B, a, \bot\alpha^{i-1}\beta)(B, a, \bot\alpha^{i-1})(B, a, \bot\alpha^{i-2}\beta) \dots (B, a, \bot\beta)$$

For IASC, consider a sequence a_i defined recursively[1] as $a_1 = 1$, $a_{i+1} = i \cdot \sum_{j=1}^{i} a_j$ and a run $\pi = \pi_{a_1}\pi_{a_2}\pi_{a_3} \dots$. For each i, we have $\sum_{j=0}^{a_1+\dots+a_i} \mathbf{c}(\pi[j]) = 3(a_1 + \dots + a_i)$ (as in the SASC case, one can assign exactly three positions to each β). Therefore, at the position $3a_i + a_{i+1}$ in π, which is in $\pi_{a_{i+1}}$ and it is the first position there with B, the value $\mathsf{ASC}(\pi, \mathbf{c}, 3a_i + a_{i+1})$) can be bounded by $\frac{3(a_1+\dots+a_i)}{3(a_1+\dots+a_i)+a_{i+1}} = \frac{3(a_1+\dots+a_i)}{(3+i)(a_1+\dots+a_i)} = \frac{3}{3+i}$. The sequence $\frac{3}{3+i}$ converges to 0, and since we only have non-negative costs, $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = 0$.

The above example uses both non-accepting states (to ensure that the stack is emptied infinitely often) and zero costs. Both are needed; we show a no-free-lunch theorem, saying that we can only have two out of three things: (1) ω-PDA with non-accepting states, (2) stack symbols with cost 0, or (3) a guarantee that IASC and SASC coincide.

▶ **Theorem 3.** *Let $\mathcal{A}$ be an ω-PDA and $\mathbf{c}$ be a stack pricing $\mathbf{c}$. If $\mathcal{A}$ has only accepting states or $\mathbf{c}$ returns only positive values, then* $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$.

Proof sketch. In the only-accepting-runs case, the theorem follows from a reduction to one-player games on WPSs with mean-payoff objectives [5]. Mean-payoff objectives are considered in two variants: mean-payoff infimum corresponding to limit infimum of partial averages and mean-payoff supremum corresponding to limit supremum of parial averages. However, it is shown in [5] that winning against one objective (say, the mean-payoff infimum objective) is equivalent to winning against the other (the mean-payoff supremum objective). From that and our reduction, we conclude that $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$. In the only-positive-values case, we prove that it is enough to consider runs with bounded size of the stack; then, we reduce the average stack cost to the regular language case, in which the results on weighted automata [2] and simple arguments show that the infimum over all runs of a weighted automaton with accepting states is realized by a run, in which partial averages converge. We conclude that $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$. ◀

[1] It can be defined explicitly as $a_i = ii!$, but the recursive definition is more convenient for us.

We conclude with a realisability theorem, stating that if $\mathsf{IASC}(\mathcal{A}, \mathbf{c})$ and $\mathsf{SASC}(\mathcal{A}, \mathbf{c})$ coincide, then there is a single run that witnesses both.

▶ **Theorem 4.** *For an ω-PDA $\mathcal{A}$ and a stack pricing $\boldsymbol{c}$ we have $\mathsf{IASC}(\mathcal{A}, \boldsymbol{c}) = \mathsf{SASC}(\mathcal{A}, \boldsymbol{c})$ iff there is a run π such that $\mathsf{IASC}(\pi, \boldsymbol{c}) = \mathsf{SASC}(\pi, \boldsymbol{c}) = \mathsf{IASC}(\mathcal{A}, \boldsymbol{c})$.*

Proof sketch. We prove that $\mathsf{SASC}(\mathcal{A}, \mathbf{c})$ is always realized, i.e., for every PDA $\mathcal{A}$ there exists π such that $\mathsf{SASC}(\pi, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$. This immediately implies the theorem. ◀

4 From Average Stack Cost to Average Letter Cost

The *average letter cost problem* takes an ω-PDA $\mathcal{A}$ and a cost function defined on letters, and asks whether there is a word in the language of $\mathcal{A}$ whose long-run average of costs of letters is below a given threshold. This section is devoted to a polynomial time reduction from the average stack cost problem to the average letter cost problem.

The reduction consists of two steps. First, we show that in the average stack cost problem, we can impose a bound B on the stack cost (which depends on the ω-PDA and the threshold). Next, we take the ω-PDA $\mathcal{A}$ from the average stack cost problem and define an ω-PDA $\mathcal{A}^M$, which recognizes words encoding the runs of $\mathcal{A}$. The words accepted by $\mathcal{A}^M$ correspond precisely to runs with stack height bounded by B and are annotated with the current stack cost along the run. These annotated costs are treated as costs of the letters, which completes the reduction.

Formally, a letter-cost function $\mathbf{lc}$ is a function from a finite alphabet of letters Σ into rationals. We assume the binary encoding of numbers. The letter-cost function extends naturally to words by $\mathbf{lc}(a_1 \dots a_n) = \mathbf{lc}(a_1) + \dots + \mathbf{lc}(a_n)$. For a finite word w, we define the average letter cost $\mathsf{avglc}(w)$ as $\frac{\mathbf{lc}(w)}{|w|}$. The average letter cost extends to infinite words as the low and the high average letter cost. For an infinite word w, we define the average low letter cost as $\mathsf{avgInflc}(w) = \liminf_{k\to\infty} \mathsf{avglc}(w[1,k])$ and the average high letter cost as $\mathsf{avgSuplc}(w) = \limsup_{k\to\infty} \mathsf{avglc}(w[1,k])$).

▶ **The** `IALC` **problem**: given an ω-PDA $\mathcal{A}$, a letter-cost function $\mathbf{lc}$, $\bowtie \in \{<, \leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that $\mathsf{avgInflc}(w) \bowtie \lambda$.

▶ **The** `SALC` **problem**: given an ω-PDA $\mathcal{A}$, a letter-cost function $\mathbf{lc}$, $\bowtie \in \{<, \leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that $\mathsf{avgSuplc}(w) \bowtie \lambda$).

In contrast to the average stack cost problem, we allow the binary encoding of numbers for $\mathbf{lc}$ and λ (a rational is encoded as a pair of integers, which are encoded in binary). The main result of this section is the following theorem:

▶ **Theorem 5.** *There are polynomial-time reductions from the* `IASC` *problem to the* `IALC` *problem and from the* `SASC` *problem to the* `SALC` *problem.*

We start the proof with auxiliary tools and lemmas.

4.1 Pumping lemma

For a run π and a position i, by $q_\pi[i]$, $a_\pi[i]$ and $u_\pi[i]$ we denote the state, letter and stack at position i of π, i.e., $(q_\pi[i], a_\pi[i], u_\pi[i]) = \pi[i]$.

Consider a run π. We define two useful functions, $\mathit{first}_\pi(i,j)$ and $\mathit{last}_\pi(i,j)$, that take a position i in π and a stack position j of the configuration $\pi[i]$ and return a position from π. Intuitively, $\pi[\mathit{first}_\pi(i,j)]$ is the configuration where the jth stack symbol in the ith configuration of π was added to the stack and $\pi[\mathit{last}_\pi(i,j)]$ is the configuration right before this stack symbol was removed from the stack. More formally, the functions first_π and last_π

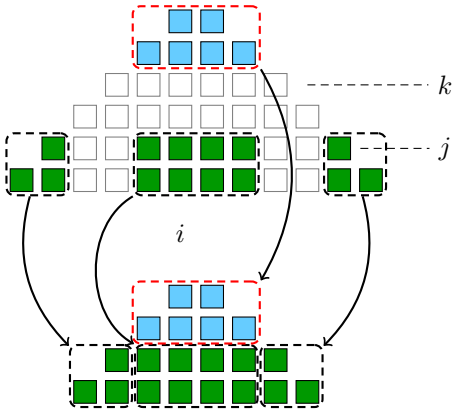


Figure 1 An example of i, j, k-contraction. The top part of the picture illustrates twelve consecutive stack contents; the sixth one is marked as i and contains two distinguished equivalent stack positions j and k. The bottom part of the picture is obtained by removing configurations 3, 4, 9 and 10 and removing stack positions 3 and 4 at positions 5-8 in the run.

are such that for each $i \in \mathbb{N}$ and each $j \in \{1, \ldots, |u_\pi[i]|\}$, $\mathit{first}_\pi(i,j)$ is a minimal number and $\mathit{last}_\pi(i,j)$ is a maximal number such that $\mathit{first}_\pi(i,j) \leq i \leq \mathit{last}_\pi(i,j)$ and all the stacks among $u_\pi[\mathit{first}_\pi(i,j)], \ldots, u_\pi[\mathit{last}_\pi](i,j)$ start with the same j stack symbols $u_1, \ldots, u_j$.

A stack position j in a configuration i is *persistent* if $\mathit{last}_\pi(i,j) = \infty$ (i.e., this symbol is never removed) and *ceasing* otherwise. We define a function *lifespan* $ls_\pi(i,j) = (\mathit{first}_\pi(i,j), \mathit{last}_\pi(i,j))$. For a finite word $w = w_1 w_2, \ldots, w_s$ let $w[l, \infty]$ denote the suffix $w_l w_{l+1} \ldots w_s$.

Assume a run π and a position $i \in \mathbb{N}$ such that $u_\pi[i] = u_1 \ldots u_n$. Two stack positions $j, k \in \{1, \ldots, n\}$ are *equivalent in* $\pi[i]$ if

- $u_j = u_k$ and they first appeared with the same symbols above, i.e., $u_\pi[\mathit{first}_\pi(i,j)][j, \infty] = u_\pi[\mathit{first}_\pi(i,k)][k, \infty]$, and
- $q_\pi[\mathit{first}_\pi(i,j)] = q_\pi[\mathit{first}_\pi(i,k)]$, and
- Either both j and k are persistent in i, or both are ceasing and then $q_\pi[\mathit{last}_\pi(i,j)] = q_\pi[\mathit{last}_\pi(i,k)]$.

Assume a run π, $i \in \mathbb{N}$ and two stack positions $j < k$ equivalent in $\pi[i]$. We define a i, j, k-contraction of π as a sequence π^C defined as follows:

- If j and k are ceasing, then $\pi^C = \pi[0, \mathit{first}_\pi(i,j)]\pi'\pi[\mathit{last}_\pi(i,j), \infty]$, where π' is the result of removing in $\pi[\mathit{first}_\pi(i,k) + 1, \mathit{last}_\pi(i,k) - 1]$ in each stack symbols at positions $j+1, j+2, \ldots, k$.
- If j and k are persistent, then $\pi^C = \pi[0, \mathit{first}_\pi(i,j)]\pi'$, where π' is the result of removing in $\pi[\mathit{first}_\pi(i,k) + 1, \infty]$ in each stack symbols at positions $j+1, j+2, \ldots, k$.

The proof of the following lemma is now straightforward.

▶ **Lemma 6.** *A contraction of an accepting run is an accepting run.*

If the stack size is at least $3|Q| \cdot |\Gamma| \cdot |\delta|$, then either there are more than $2|\delta|$ persistent positions on the stack with the same stack symbol or more than $2|Q||\delta|$ ceasing positions with the same stack symbols; in both cases, one can always pick three equivalent positions among them. We state this observation as a lemma.

▶ **Lemma 7.** *Among any $3|Q| \cdot |\Gamma| \cdot |\delta|$ stack positions in any configuration $\pi[i]$ there are three stack positions pairwise equivalent in $\pi[i]$.*

4.2 The bounded stack cost property

We show the bounded stack cost property for the IASC and SASC problem.

▶ **Lemma 8.** *Assume an ω-PDA $\mathcal{A}$, stack pricing $\boldsymbol{c}$, $\bowtie \in \{<, \leq\}$, $\lambda \in \mathbb{Q}$ and an accepting run π such that $\mathsf{IASC}(\pi) \bowtie \lambda$. There is an accepting run π' such that $\mathsf{IASC}(\pi') \bowtie \lambda$ and for each i, $\boldsymbol{c}(\pi[i]) \leq \max_{s\in\Gamma} \boldsymbol{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$. The same holds for* SASC.

Proof sketch. We first show the proof for IASC and $\bowtie = \leq$. Assume an ω-PDA $\mathcal{A}$, stack pricing $\mathbf{c}$, $\lambda \in \mathbb{Q}$ and an accepting run π such that $\mathsf{IASC}(\pi) \leq \lambda$.

Let i be the smallest number such that $\mathbf{c}(\pi[j]) \leq i$ for infinitely many j. Clearly $i \leq \lambda$ since $\mathsf{IASC}(\pi) \leq \lambda$. If there are only finitely many positions where the stack cost exceeded $\max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$, then for every such a position i we can find, by Lemma 7, two equivalent stack positions $j < k$ and obtain the i, j, k-contraction of the run. We repeat it until the cost reaches the desired bound. Since we repeat this only finitely many times for the whole run, the obtained run π' is accepting and $\mathsf{IASC}(\pi') \leq \lambda$.

For the rest of this proof, we focus on the case where there are infinitely many positions with costly stack. There are two new challenges now: we need to make sure to preserve infinitely many accepting states, and guarantee that IASC stays within desired bound.

To preserve infinitely many accepting states, we decompose π as $\pi_1^{ok}\pi_1\pi_2^{ok}\pi_2\pi_3^{ok}\dots$ such that every positions in π_i has stack cost exceeding λ and all the positions of π_i^{ok} cost at most λ. Our goal is to define a new run π' that is obtained from π by contracting some of the runs among $\pi_1, \pi_2, \dots$. To preserve the acceptance condition, we mark one configuration with an accepting state in each π_i that has such a configuration and guarantee that this state will be retained.

Let $\pi' = \pi_1^{ok}\pi_1'\pi_2^{ok}\pi_2'\pi_3^{ok}\dots$, where for each i, we define π_i' starting from π_i, and then by repeating the following procedure as long as needed. For any j such that $\mathbf{c}(\pi_i'[j]) > \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$, there are at least $3|Q| \cdot |\Gamma| \cdot |\delta|$ stack positions k whose symbols have positive cost and lifespan is within π', as the total cost of the remaining stack position is bounded by λ. Among them, we can find three pairwise equivalent stack positions, from which one can choose two positions k, l such that the j, k, l-contraction of π_i' retains the marked accepting state. We set π_i' to be the contraction.

For each position i in π', we define its origin $o(i)$ as the position in π from which i originates (o is a monotonic function). We argue that if $\mathsf{ASC}(\pi, \mathbf{c}, i) \leq \lambda$, then $\mathsf{ASC}(\pi', \mathbf{c}, o(i)) \leq \lambda$; the proof is based on the fact that we only remove or alter positions where the cost is greater than λ (see [7] for details). It follows that $\mathsf{IASC}(\pi') \leq \mathsf{IASC}(\pi)$, as required.

For the strict inequality assume that $\mathsf{IASC}(\pi) < \lambda$. Then for some $\epsilon > 0$, we have $\mathsf{IASC}(\pi) \leq \lambda - \epsilon$. By the above reasoning, there exists π' such that $\mathsf{IASC}(\pi') \leq \lambda - \epsilon$ and for each i, $\mathbf{c}(\pi[i]) \leq \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$. Then, $\mathsf{IASC}(\pi') < \lambda$.

The case of SASC uses the same construction, but the reasoning now is slightly more technical as we have to argue that all the subsequences of π' have the cost less than λ, but the idea is exactly the same, so we skip it here. ◀

We now prove Theorem 5.

Proof. We show the reduction of IASC to IALC. The reduction is through a construction of a *meta-automaton* defined below. Given an instance $I = \langle \mathcal{A}, \mathbf{c}, \bowtie, \lambda \rangle$ of the IASC problem, we define a *meta-automaton* $\mathcal{A}^M$ for I as an ω-PDA that recognizes the language of infinite words corresponding to accepting runs of $\mathcal{A}$. Formally, let $N = \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$. The ω-PDA $\mathcal{A}^M$ works over the alphabet $\delta \times \{0, \dots, N\}$, where δ is the transition relation of $\mathcal{A}$, and $\mathcal{A}^M$ accepts all words w such that (1) w encodes an accepting run π^w of $\mathcal{A}$, and

(2) the stack cost of π at position i is encoded as the second component of $w[i]$. In particular, (2) implies that the meta-automaton accepts words that correspond to runs of $\mathcal{A}$ in which the stack cost is bounded by N at every position.

To build a meta-automaton we need to track the current stack cost. However, even a finite-state automaton can store in its states the current stack cost, which belongs to $\{0,\dots,N\}$ and update it based in the current symbol being pushed to or popped from the stack. Therefore, a meta-automaton can be constructed in polynomial time.

Consider a letter-cost function defined over $\delta \times \{0,\dots,N\}$ such that $\mathbf{lc}(t,x) = x$, i.e., the letter cost of a transition is the current stack cost. Observe that each partial averages of stack cost in a run π coincides with the corresponding partial average of letter costs in the corresponding word. Therefore, if the instance $(\mathcal{A}^M, \mathbf{lc}, \bowtie, \lambda)$ of the IALC problem is solved by a word w, then the corresponding run π^w of $\mathcal{A}$ is a solution of I. Conversely, if I has a solution π, then by Lemma 8 it has a solution π', in which all stack costs are bounded by N. Then, there is a word w' accepted by $\mathcal{A}^M$ corresponding to the run π'. Observe that w' is a solution of the instance $(\mathcal{A}^M, \mathbf{lc}, \bowtie, \lambda)$ of the IALC problem.

The same construction gives us the reduction from SASC to SALC. The prove of correctness is the same as we only need to use different variant of Lemma 8. ◀

5 The average letter cost problem

We prove that the average letter cost problem can be solved in polynomial time. In Section 5.1 we study the finite-word variants of the average letter cost problem. Next, we use finite-word results to solve the average letter cost problem over infinite word (Section 5.2). We supplement this section with the comparison of the average letter cost problem and one-player games on WPSs with conjunctions of mean-payoff and Büchi objectives.

5.1 Average letter cost over finite words

We are interested in the average letter cost over all (finite) words accepted by a given PDA.

▶ **The** avglc **problem**: given a letter-cost function $\mathbf{lc}$, a PDA $\mathcal{A}$, $\bowtie \in \{<, \leq\}$ and a threshold λ, decide whether $\inf_{w\in\mathcal{L}(\mathcal{A})} \mathsf{avglc}(w) \bowtie \lambda$.

To solve this problem, we first discuss how to compute the infimum of $\mathbf{lc}(w)$ over all words accepted by $\mathcal{A}$, i.e., $\inf_{w\in\mathcal{L}(\mathcal{A})} \mathbf{lc}(w)$. Next, we solve the average letter cost problem by computing $\inf_{w\in\mathcal{L}(\mathcal{A})} \mathbf{lc}^\lambda(w)$ for a modified letter cost function $\mathbf{lc}^\lambda$.

▶ **Lemma 9.** *Given a PDA $\mathcal{A}$ and a letter-cost function $\mathbf{lc}$, we can compute $\inf_{w\in\mathcal{L}(\mathcal{A})} \mathbf{lc}(w)$, the infimum of $\mathbf{lc}(w)$ over all words accepted by $\mathcal{A}$, in polynomial time in $\mathcal{A}$.*

▶ Remark. The values of the letter-cost function in Lemma 9 can be represented in binary.

Proof sketch. To compute $\inf_{w\in\mathcal{L}(\mathcal{A})} \mathbf{lc}(w)$, we transform $\mathcal{A}$ to a CFG G generating the same language. Then, we adapt the classic algorithm for checking the emptiness of the language generated by a CFG [6]. The algorithm from [6] marks iteratively non-terminals that derive some words. It starts by marking non-terminals that derive a single letter. Then it takes $|G|$ iterations of a loop, in which it applies all the rules of G, and marks non-terminals that derive marked non-terminals. Here, we associate with each non-terminal A, a variable v_A storing the minimal value of $\mathbf{lc}(w)$ for w derivable from A in G. We update values of v_A in each iteration, by putting $v_A = \min(v_A, v_B + v_C)$ for every rule $A \to BC$. The algorithm terminates if (1) there is an iteration, in which no variable has changed or (2) after $|G|+1$ iterations. In the first case, we return v_S, the computed value for the start symbol. In the

second case, we observe that no further iterations are necessary as there exists a derivation $B \to_G^* v_L B v_R$ with $\mathbf{lc}(v_L v_R) < 0$, and hence $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathbf{lc}(w) = -\infty$. ◀

Using Lemma 9, we can solve the average letter cost problem in the finite word case.

▶ **Lemma 10.** *The* avg**lc** *problem can be solved in polynomial time.*

Proof sketch. First, if $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathsf{avg}\mathbf{lc}(w) < \lambda$, then there is a word w with $\mathsf{avg}\mathbf{lc}(w) < \lambda$. Observe that the average of $a_1, \ldots, a_n$ is less than lambda if and only if the sum of $(a_1 - \lambda), \ldots, (a_n - \lambda)$ is less than 0. Therefore, to check existence of w with $\mathsf{avg}\mathbf{lc}(w) < \lambda$, we define $\mathbf{lc}^\lambda$ by substructing λ from each value of **lc** and apply Lemma 9 to check existence of w with $\mathbf{lc}^\lambda(w) < 0$. The case of the non-strict inequality is more difficult as the infimum need not be realized. However , we consider a CFG G generating the language of $\mathcal{A}$ and we show that $\inf_{u \in \mathcal{L}(\mathcal{A})} \mathsf{avg}\mathbf{lc}(u) \leq \lambda$ if and only if (1) there exists $u \in \mathcal{L}(\mathcal{A})$ with $\mathsf{avg}\mathbf{lc}(u) \leq \lambda$, or (2) there is a non-terminal A such that $A \to_G^* u_L A u_R$ and $\mathsf{avg}\mathbf{lc}(u_L u_R) \leq \lambda$. Both conditions can be checked in polynomial time using the letter-cost function $\mathbf{lc}^\lambda$ and Lemma 9. ◀

5.2 Average letter cost over infinite words

In this section, we discuss the average letter cost problem in the infinite-word case. We begin with the average letter cost problem with the limit supremum of partial averages.

▶ **Lemma 11.** *The* SALC *problem can be decided in polynomial time.*

Proof sketch. We reduce the problem to the finite-word case and use Lemma 10. Consider an instance of the SALC problem consisting of an ω-PDA $\mathcal{A}$, letter-cost function **lc**, $\bowtie \in \{<, \leq\}$ and λ. Any context-free omega language $\mathcal{L}$ can be presented as $\mathcal{L} = \bigcup_{1 \leq i \leq k} V_i (U_i)^\omega$, where $V_1, U_1, \ldots, V_k, U_k$ are non-empty finite-word context-free languages. We can look for w in each $V_i(U_i)^\omega$ separately. Then, we show that there exists a word $w \in V_i(U_i)^\omega$ such that $\mathsf{avgSup}\mathbf{lc}(w) \bowtie \lambda$ if and only if $\inf_{u \in U_i} \mathsf{avg}\mathbf{lc}(u_i) \bowtie \lambda$. The latter condition can be checked in polynomial time (Lemma 10). ◀

Now, we consider the average letter cost problem with the limit infimum of partial averages. We again reduce it to the finite-word case, but now the reduction is not as straightforward. The following example explains the main difficulty.

▶ **Example 12.** Consider $\Sigma = \{0, 2\}$, and the letter-cost function **lc**, which simply returns the value of a letter, i.e., for $x \in \Sigma$ we have $\mathbf{lc}(x) = x$. Now, let $\mathcal{A}$ be an ω-PDA accepting the language U_0^ω, where $U_0 = \{0^n 2^n \mid n \in \mathbb{N}\}$. Observe that for every $w \in \mathcal{L}(\mathcal{A})$ we have $\mathsf{avgSup}\mathbf{lc}(w) = 1$. However, for a word $w_0 = 020^2 2^2 \ldots 0^{2^{n^2}} 2^{2^{n^2}} \ldots$ we observe that $\mathsf{avg}\mathbf{lc}(020^2 2^2 \ldots 0^{2^{n^2}}) < 2^{(n-1)^2 + 1 - n^2} = 2^{-2(n-1)}$, and hence $\mathsf{avgInf}\mathbf{lc}(w_0) = 0$. Therefore, $\mathsf{avgSup}\mathbf{lc}(w_0) \neq \mathsf{avgInf}\mathbf{lc}(w_0)$. We conclude that for a language of the form U^ω, knowing average letter costs of words in U is insufficient to decide whether there is a word w with $\mathsf{avgInf}\mathbf{lc}(w) \leq \lambda$. Still, in the following we show how to decide $\mathsf{avgInf}\mathbf{lc}(w) \leq \lambda$ by examining the structure of U.

▶ **Lemma 13.** *The* IALC *problem can be decided in polynomial time.*

Proof sketch. Again we represent the language of an ω-PDA $\mathcal{A}$ as $\mathcal{L} = \bigcup_{1 \leq i \leq k} V_i (U_i)^\omega$, where $V_1, U_1, \ldots, V_k, U_k$ are non-empty finite-word context-free languages and look for w in each $V_i(U_i)^\omega$ separately. Let G_i be a CFG generating the language U_i. We assume that G_i is pruned, i.e., every non-terminal occurs in some derivation of some word. For

$\bowtie \in \{<, \leq\}$, we show that there exists a word $w \in V_i(U_i)^\omega$ such that $\mathsf{avgInflc}(w) \bowtie \lambda$ if and only if (1) $\inf_{u \in U_i} \mathsf{avglc}(u_i) \bowtie \lambda$ or (2) there exists a non-terminal A in G_i, such that $\inf\{\mathsf{avglc}(u_L) \mid A \to^*_{G_i} u_L A u_R\} \bowtie \lambda$. Both conditions can be checked in polynomial time using Lemma 10. Condition (1) is inherited from the $\mathsf{avgSuplc}$. For condition (2), observe that $\mathsf{avgInflc}(w) \leq \lambda$ if there is a subsequence of partial averages that converges to a value at most λ. For a word $vu_1u_2 \ldots \in V_i(U_i)^\omega$, the subsequence from the limit infimum may pick only positions inside words $u_1, u_2, \ldots$ (as in Example 12), and the subsequence of partial averages at boundaries of words may converge to a higher value. Condition (2) covers this case. In Example 12, U_0 is generated by a grammar $S \to 0S2, S \to \epsilon$ and observe that this grammar satisfies condition (2) with $\lambda = 0$, i.e., for $S \to 0S2$ we have $\mathsf{avglc}(0) \leq 0$. ◀

5.3 Weighted pushdown systems with fairness

We briefly discuss the connection between the average letter cost problem and one-player games on WPSs with conjunctions of mean-payoff and Büchi objectives.

A *WPS-game* consists of a WPS $\mathcal{P} = (\mathcal{A}, \mathrm{wt})$ and a *game objective*. In each WPS-game, the only player plays infinitely many rounds selecting consecutive transitions in order to obtain a run satisfying given objectives. A game objective is a conjunction of a *mean-payoff* objective and a Büchi objective, defined as follows.

A *mean-payoff* objective is of the form $\mathsf{LimAvgInf}(\pi) \bowtie \lambda$ or $\mathsf{LimAvgSup}(\pi) \bowtie \lambda$, where $\bowtie \in \{<, \leq\}$ and $\lambda \in \mathbb{Q}$. The interpretation of such an objective is as follows: each play constructs a run π of $\mathcal{P}$ and the *cost sequence* $\mathbf{wt}(\pi)$ of π which is the sequence of costs of the transitions of π. We interpret $\mathsf{LimAvgInf}(\pi)$ as $\liminf_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} \mathbf{wt}(\pi)[i]$ and $\mathsf{LimAvgSup}(\pi)$ as $\limsup_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} \mathbf{wt}(\pi)[i]$, and we say that a mean-payoff objective is satisfied if its inequality holds. A Büchi objective is a set of states; it is satisfied by π if π visits some state from Q_F infinitely often.

To *solve a WPS-game* is to determine whether the player can construct a run that satisfies all the game objectives. The following theorem extends [5, Theorem 1] by adding Büchi objectives.

▶ **Theorem 14.** *Each WPS-game can be solved in polynomial time.*

The proof follows from a reduction to the average letter cost problem that encodes the transitions costs in corresponding letter costs. A converse polynomial-time reduction is also possible; in this case, we encode letter costs in transition costs.

6 Average Response Time Example

In this section, we use the ASC problem to compute a variant of the average response time property [3]. In this variant, there are two agents: a *client* and a *server*. A client can state a request, which is later granted or rejected by the server. Requests are dealt with on the first-come, first-served basis, but not immediately — the server may need some time to issue a grant. We assume that both client and server are modeled as systems with finitely many states and can check whether the number of pending requests at a given moment is zero. We also assume a fairness condition stating that there are infinitely many requests and grants.

A trace of such system is a word over the alphabet $\{r, g, \#\}$, where r denotes a new request, g denotes a grant and $\#$ denotes a null instruction. We are interested in bounding the minimal possible average response time of such a model . In other words, we are interested

checking, for a given λ and a model, whether

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} a_i < \lambda \tag{1}$$

for some computation of the model in which the ith request was realised after a_i steps of computations (our technique works for lim sup as well, but we focus on lim inf).

Feasibility study. To apply our technique, we need to overcome two main difficulties. First, our technique only works for stacks, but requests are handled in a queue manner. In general, non-emptiness of automata with queue is undecidable. Second, the denominator in (1) refers only to the number of requests, not the number of positions in words (they may differ because of the letter #).

Dealing with queues. We abstract the counter to a stack over a unary alphabet $\{P\}$ whose size equals the value of the counter in the straightforward way.

We claim that there is a run satisfying (1) if and only if there is a run satisfying

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) < \lambda \tag{2}$$

where G_n denotes the position of the nth grant in the run (we assume that each grant correspond to a request).

We present the main idea (see [7] for details). Consider a position in a run n where there are no pending requests; then, the total waiting time of all requests up to this position is equal to the sum of the number of waiting processes in each position up to n. At a position with unfulfilled requests, this is no longer guaranteed, as the pending processes may have some waiting time in the future. However, it can be shown that a run for (2) can be chosen in a way that guarantees that the difference between the two numbers is bounded by some constant, and therefore can be neglected in the lim inf.

Selected positions. We argue that (2) is equivalent to

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda \tag{3}$$

where $\mathbf{c}^*(\pi[i])$ equals $\mathbf{c}(\pi[i])$ if the position i corresponds to a grant and $\mathbf{c}(\pi[i]) + \lambda$ otherwise.

Observe that $\frac{1}{n}\sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) < \lambda$ iff $\sum_{i=1}^{g_n} \mathbf{c}(\pi[i]) - n\lambda + g_n\lambda < g_n\lambda$ iff $\frac{1}{g_n}\sum_{i=1}^{G_n} \mathbf{c}^*(\pi[i]) < \lambda$. The last equivalence follows from the fact that there are n grants, and so $\sum_{i=1}^{G_n} \mathbf{c}^*(\pi[i]) = \sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) + (G_n - n)\lambda$.

If (2), then there is an infinite sequence of positions where $\frac{1}{n}\sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) < \lambda$, and by the above reasoning each position in this sequence satisfies $\frac{1}{G_n}\sum_{i=1}^{G_n} \mathbf{c}^*(\pi[i]) < \lambda$. This means that (2) implies (3). The converse if also true. To see this, observe that if at a position $n > 0$ that does not correspond to a grant we have $\frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda$, then also $\frac{1}{n-1}\sum_{i=1}^{n-1} \mathbf{c}^*(\pi[i]) < \lambda$ as $\mathbf{c}^*(\pi[n]) \geq \lambda$. If we have an infinite sequence of positions with $\frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda$ and infinitely many grants, we can select an infinite sequence of positions n corresponding to grants such that $\frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda$.

Putting it all together. From the above consideration, we know that (1) if and only if (3). Therefore, to verify (1), we modify the automaton as follows: we add an additional stack symbol $\bullet$ of weight $\lambda + 1$ that can only appear at the top of the stack. We modify the transition function to stipulate that whenever the automaton is in a position that does not correspond to a grant, then the topmost symbol is $\bullet$. By our results, checking whether there is a run with the average stack cost less than λ (and therefore whether the average waiting time is less than λ) can be done in polynomial time.

References

1. Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G Larsen, and Simon Laursen. Average-energy games. *Acta Informatica*, pages 1–37, 2015.
2. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.
3. Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. In *LICS 2015*, pages 725–737, 2015.
4. Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Bidirectional nested weighted automata. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.5.
5. Krishnendu Chatterjee and Yaron Velner. Mean-payoff pushdown games. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 195–204. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.30.
6. John E. Hopcroft and Jefferey D. Ullman. *Introduction to Automata Theory, Languages, and Comput ation.* Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.
7. Jakub Michaliszyn and Jan Otop. Average stack cost of buechi pushdown automata. *CoRR*, abs/1710.04490, 2017. URL: http://arxiv.org/abs/1710.04490.
8. Yasuhiko Minamide. Weighted pushdown systems with indexed weight domains. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 230–244. Springer, 2013.
9. Thomas W. Reps, Akash Lal, and Nicholas Kidd. Program analysis using weighted pushdown systems. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 23–51. Springer, 2007. doi:10.1007/978-3-540-77050-3_4.
10. Thomas W. Reps, Stefan Schwoon, Somesh Jha, and David Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005. doi:10.1016/j.scico.2005.02.009.

Querying Best Paths in Graph Databases*†

Jakub Michaliszyn[1], Jan Otop[2], and Piotr Wieczorek[3]

1 **Institute of Computer Science, University of Wrocław, Poland**
jmi@cs.uni.wroc.pl
2 **Institute of Computer Science, University of Wrocław, Poland**
jotop@cs.uni.wroc.pl
3 **Institute of Computer Science, University of Wrocław, Poland**
piotrek@cs.uni.wroc.pl

Abstract

Querying graph databases has recently received much attention. We propose a new approach to this problem, which balances competing goals of expressive power, language clarity and computational complexity. A distinctive feature of our approach is the ability to express properties of minimal (e.g. shortest) and maximal (e.g. most valuable) paths satisfying given criteria. To express complex properties in a modular way, we introduce labelling-generating ontologies. The resulting formalism is computationally attractive – queries can be answered in non-deterministic logarithmic space in the size of the database.

1998 ACM Subject Classification H.2.4 Query processing, H.2.3 Query Languages

Keywords and phrases graph databases, queries, aggregation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.43

1 Introduction

Graphs are one of the most natural representations of data in a number of important applications such as modelling transport networks, social networks, technological networks (see surveys [2, 39, 5]). The main strength of graph representations is the possibility to naturally represent not only the data itself, but also the links among data. Effective search and analysis of graphs is an important factor in reasoning performed in various AI tasks. This motivates the study of query formalisms for graph databases, which are capable of expressing properties of paths.

One of the most challenging problems of recent years is to process big data, typically too large to be stored in the modern computers' memory. This stimulates a strong interest in algorithms working in logarithmic space w.r.t. the size of the database (data complexity) [11, 4, 7]. At the same time, even checking whether there is a path between two given nodes is already NL-complete, so NL is the best complexity for any expressive graph query language.

Our contribution. We propose a new approach to writing queries for graph databases, in which labelling-generating ontologies are first-class citizens. It can be integrated with many existing query formalisms. However, in order to make the presentation clear we introduce the concept by defining a new language OPRA. OPRA features NL-data complexity, good expressive power and a modular structure. The expressive power of OPRA strictly subsumes

* This work has been supported by Polish National Science Center grant UMO-2014/15/D/ST6/00719.
† The full version of this work is available as [33], https://arxiv.org/abs/1710.04419.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 43; pp. 43:1–43:15

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

the expressive power of popular existing formalisms with same complexity (see Fig. 1). Distinctive properties expressible in OPRA are based on aggregation of data values along paths and computation of extremal values among aggregated data. One example of such a property is "p is a path from s to t that has both the minimal weight and the minimal length among all paths from s to t".

To ease the presentation, we define OPRA in two steps. First, we define the language PRA, whose main components are three types of constraints: P*ath*, R*egular* and A*rithmetical*. We use path constraints to specify endpoints of graph paths; the other constraints only specify properties of paths. Regular constraints specify paths using regular expressions, adapted to deal with multiple paths and infinite alphabets. Arithmetical constraints compare linear combinations of aggregated values, i.e., values of *labels* accumulated along whole paths.

The language PRA can only aggregate and compare the values of labelling functions already defined in the graph. The properties we are interested in often require performing some arithmetical operations on the labellings, either simple (taking a linear combination) or complicated (taking minimum, maximum, or even computing some subquery). Such operations are often nested inside regular expressions (as in LARE [23]) making queries unnecessarily complicated. Instead, similarly as in [3] we specify such operations in a modular way as *ontologies*. This leads to the language OPRA, which comprises Ontologies and PRA. In our approach all knowledge on graph nodes is encoded by labellings, and our ontologies also are defined as the auxiliary labellings. For example, having a labelling child(x, y) stating that x is a child of y, we can define a labelling descendant(x, y) stating that x is a descendant of y. Such labellings can be computed on-the-fly during the query evaluation.

Related work. *Regular Path Queries (RPQs)* [15, 12] are usually used as a basic construction for graph querying. RPQs are of the form $x \to^{\pi} y \wedge \pi \in L(e)$ where e is a (standard) regular expression. Such queries return pairs of nodes (v, v') connected by a path π such that the labelling of π forms a word from $L(e)$. *Conjunctive Regular Path Queries (CRPQs)* are the closure of RPQs under conjunction and existential quantification [14, 32]. Barcelo et al., [7] introduced *extended CRPQs (ECRPQs)* that can compare tuples of paths by *regular relations* [19, 22]. Examples of such relations are path equality, length comparisons, prefix (i.e., a path is a prefix of another path) and fixed edit distance. Regular relations on tuples of paths can be defined by the standard regular expressions over alphabet of tuples of edge symbols.

Graph nodes often store *data values* from an infinite alphabet. In such graphs, paths are interleaved sequences of data values and edge labels. This is closely related to *data words* studied in XML context [34, 18, 37, 10]. Data complexity of query answering for most of the formalisms for data words is NP-hard [30]. This is not, however, the case for *register automata* [26], which inspired Libkin and Vrgoč to define *Regular Queries with Memory (RQMs)* [30]. RQMs are again of the form $x \to^{\pi} y \wedge \pi \in L(e)$. However, e is now *Regular Expression with Memory (REM)*. REMs can store in a register the data value at the current position and test its equality with other values already stored in registers. Register Logic [6] is, essentially, the language of REMs closed under Boolean combinations, node, path and register-assignment quantification. It allows for comparing data values in different paths. The positive fragment of Register Logic, RL^{+}, has data complexity in NL, even when REMs can be nested using a branching operator. Walk Logic [25] extends FO with path quantification and equality tests of data values on paths. Query answering for WL is decidable but its data complexity is not elementary [6]. LARE [23] is a query language that can existentially quantify nodes and paths, and check relationship between many paths. Path relationships are defined by regular expressions with registers that allow for various arithmetic operations on registers.

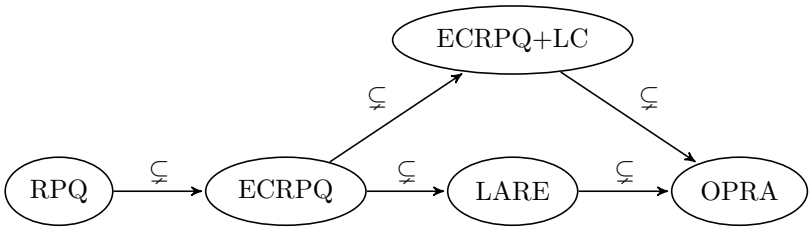


Figure 1 Comparison between different query languages.

Aggregation. Ability to use aggregate functions such as sum, average or count is a fundamental mechanism in database systems. Klug [28] extended the relational algebra and calculus with aggregate functions and proved their equivalence. Early graph query languages G^+ [16] or GraphLog [14, 13] can aggregate data values. Consens and Mendelzon [13] studied *path summarization*, i.e., summarizing information along paths in graphs. They assumed natural numbers in their data model and allowed to aggregate summarization results. In order to achieve good complexity (in the class NC) they allowed aggregate and summing operators that form a closed semiring. Other examples of aggregation can be found in [39].

Summing vectors of numbers along graph paths have been already studied in the context of various formalisms based on automata or regular expressions and lead to a number of proposals that have combined complexity in PSPACE and data complexity in NL. Kopczyński and To [29] have shown that *Parikh images* (i.e., vectors of letter counts) for the usual finite automata can be expressed using a union of linear sets that is polynomial in the size of the automaton and exponential in the alphabet size (the alphabet size, in our context, corresponds to the dimension of vectors). Barcelo et al. [7] extended ECRPQs with linear constraints on the numbers of edge labels counts along paths. They expressed the constraints using reversal-bounded counter machines, translated further to Presburger arithmetic formulas of a polynomial size and evaluate them using techniques from [29, 36].

Figueira and Libkin [20] studied *Parikh automata* introduced in [27]. These are finite automata that additionally store a vector of *counters* in $\mathbb{N}^k$. Each transition specifies also a vector of natural numbers. While moving along graph paths according to a transition the automaton adds this transition's vector to the vector of counters. The automaton accepts if the computed vector of counters is in a given semilinear set in $\mathbb{N}^k$. Also a variant of regular expressions capturing the power of these automata is shown. This model has been used to define a family of variants of CRPQs that can compare tuples of paths using *synchronization languages* [21]. This is a relaxation of regularity condition for relations on paths of ECRPQs and leads to more expressive formalisms with data complexity still in NL. These formalisms are incomparable to ours since they can express non-regular relations on paths like suffix but cannot express properties of data values, nodes' degrees or extrema.

Cypher [38] is a practical query language implemented in the graph database Neo4j. It uses *property graphs* as its data model. These are graphs with labelled nodes and edges, but edges and nodes can also store attribute values for a set of *properties*. `MATCH` clause of Cypher queries allows for specifying graph patterns that depend on nodes' and edges' labels as well as on their properties values. Cypher does not allow full regular expressions however graph patterns can contain transitive closure over a single label. More on Cypher can be found in a survey [2].

RDF [17] is a W3C standard that allows encoding of the content on the Web in a form of a set of *triples* representing an edge-labelled graph. Each triple consists of the subject s, the predicate p, and the object o that are resource identifiers (URI's), and represents an edge

from s to o labelled by p. Interestingly, the middle element, p, may play the role of the first or the third element of another triple. Our formalism OPRA allows to operate directly on RDF without any complex graph encoding, by using a ternary labelling representing RDF triples. This allows for convenient navigation by regular expressions in which also the middle element of a triple can serve as the source or the target of a single navigation step (cf. [31]). The standard query formalism for RDF is SPARQL [35, 24]. It implements *property paths* which are RPQs extended with inverses and limited form of negation (see survey [2]).

2 Language OPRA

Various kinds of data graphs are possible and presented in the literature. The differences typically lie in the way the elements of graphs are labelled – both nodes and edges may be labelled by finite or infinite alphabets, which may have some inner structure. Here, we choose a general approach in which a *labelled graph*, or simply a graph, is a tuple consisting of a finite number of *nodes* V and a number of labelling functions $\lambda : V^l \to \mathbb{Z} \cup \{-\infty, \infty\}$ assigning integers to vectors of nodes of some fixed size. While edges are not explicitly mentioned, if needed, one can consider an *edge* labelling λ_E such that $\lambda_E(v, v')$ is 1 if there is an edge from v to v' and it is 0 otherwise. More sophisticated edges, e.g., with integer labels, may be handled by means of standard embedding, defined in Section 4. For convenience, we assume that the set of nodes always contains a distinguished node $\square$ – we use it as a "sink node", to avoid problems with paths of different lengths.

A *path* is a sequence of nodes. For a path $p = v_1 \dots v_k$, by $p[i]$ we denote its i-th element, v_i, if $i \leq k$, and $\square$ otherwise.

2.1 Basic constructs

We first define the language PRA, which is the core of the language OPRA. The queries of PRA are of the form

```
MATCH NODES x⃗, PATHS π⃗
SUCH THAT Path_constraints
WHERE Regular_constraints
HAVING Arithmetical_constraints
```

where $\vec{x}$ are free node variables, $\vec{\pi}$ are free path variables, `Path_constraints` is a conjunction of path constraints, `Regular_constraints` is a conjunction of *regular constraints* and, finally, `Arithmetical_constraints` is a conjunction of *arithmetical constraints*, as defined below. The constraints may contain variables not listed in the `MATCH` clause (which are then existentially quantified).

Path constraints. Path constraints are expressions of the form $x_s \to^{\pi} x_t$, where x_s, x_t are node variables and π is a path variable, satisfied if π is a sequence of nodes starting from x_s and ending in x_t.

Regular constraints. The main building blocks of regular constraints are *node constraints*. Syntactically, a k-node constraint is an expression containing free node variables $@_1, @'_1, \dots, @_k, @'_k$ and of the form $X \sim X'$, where $\sim \in \{\leq, <, =\}$ and each of X, X' is an integer constant or a labelling function λ_i applied to some of the free variables.

A k-node constraint for a regular constraint over k paths may be seen as a function that takes a vector containing two nodes of each path: a *current node* (represented by $@_i$) and a *next node* (represented by $@'_i$), and returns a Boolean value. The semantics is given by

applying the appropriate labelling functions to the nodes given as an input and comparing the value according to the $\sim$ symbol.

A regular constraint $R(\pi_1, \ldots, \pi_k)$ is syntactically a regular expression over an infinite alphabet consisting of all the possible k-node constraints. Assume $p_1, \ldots, p_k$ are paths and let $w_1 \ldots w_s$ be the word such that $s = \max(|p_1|, \ldots, |p_k|)$ and each $w_i \in V^{2k}$ is defined as $w_i = (p_1[i], p_1[i+1], \ldots, p_k[i], p_k[i+1])$, i.e., it is a vector consisting of i-th and $i+1$-th node of each path (or can be substituted by $\Box$ if not present). We say that $p_1, \ldots, p_k$ *satisfy* R, denoted as $R(p_1, \ldots, p_k)$, if the $w_1 \ldots w_s$ belongs to the language $L^G(R)$, defined inductively in the usual manner:

- $L^G(R)$, where R is a node constraint, is defined as a set of vectors of length $2k$ for which the constraint R returns true.
- $L^G(R \cdot R') = \{a \cdot b \mid a \in L^G(R) \wedge b \in L^G(R')\}$.
- $L^G(R + R')$ is the union of $L^G(R)$ and $L^G(R')$.
- $L^G(R^*)$ is the Kleene-star closure of $L^G(R)$.

Note that acccording to the definitions above, the variables $@_1, @'_1, \ldots, @_k, @'_k$ in a regular constraint $R(\pi_1, \ldots, \pi_k)$ always refer to the nodes of the paths $\pi_1, \ldots, \pi_k$, e.g., $@_4$ refers to the current node of the path π_4 and $@'_2$ refers to the next node of the path π_2. In order not to mix the variables with ordinary ones we disallow to use them in any other context.

Arithmetical constraints. An arithmetical constraint is an inequality $c_1\Lambda_1 + \ldots + c_j\Lambda_j \leq c_0$, where $c_0, \ldots, c_j$ are integer constants and each Λ_ℓ is an expression of the form $\lambda_i[\pi_{i_1}, \ldots, \pi_{i_k}]$. The value of $\lambda_i[\pi_{i_1}, \ldots, \pi_{i_k}]$ over paths $p_1, \ldots, p_n$ is defined as the sum $\sum_{i=1}^{s} \lambda_i(p_{i_1}[i], \ldots, p_{i_k}[i])$, where $s = \max\{|p_1|, \ldots, |p_k|\}$, i.e., the sum of the labelling for vectors of nodes on corresponding positions of all paths. Paths $\vec{p}$ satisfy the arithmetical expression $c_1\Lambda_1 + \ldots + c_j\Lambda_j \leq c_0$ if the value of the left hand side, with $\vec{\pi}$ instantiated to $\vec{p}$, is less than or equal to c_0.

Query semantics. Let $Q(\vec{x}, \vec{p})$ be a PRA query, and $\vec{x}'$ and $\vec{\pi}'$ be node and path variables in Q that are not listed as free. We say that nodes $\vec{v}$ and paths $\vec{p}$ (of some graph G) satisfy Q, denoted as $Q(\vec{v}, \vec{p})$, if and only if there exist nodes $\vec{v}'$ and paths $\vec{p}'$ such that the instantiation $\vec{x} = \vec{v}$, $\vec{x}' = \vec{v}'$, $\vec{\pi} = \vec{p}$ and $\vec{\pi}' = \vec{p}'$ satisfies all constraints in Q.

2.2 Auxiliary labelling

We introduce a way of defining auxiliary labellings of graphs, which are defined based on existing graph labellings and its structure. The ability to define auxiliary labellings significantly extends the expressive power of the language. The essential property of auxiliary labellings is that their values do not need to be stored in the database, which would require polynomial memory, but can be computed *on demand*. An auxiliary labelling may be seen as an *ontology* or a *view*.

We assume a set $\mathcal{F}$ of *fundamental functions* $f : (\mathbb{Z} \cup \{-\infty, \infty\})^* \to \mathbb{Z} \cup \{-\infty, \infty\}$ consisting of *aggregate functions* maximum Max, minimum Min, counting Count, summation Sum, and *binary* functions $+$, $-$, $\cdot$ and $\leq$ (assuming 0 for false and 1 for true, and that these functions return 0 if the number of inputs is not two). $\mathcal{F}$ can be extended, if needed, by any functions computable by a non-deterministic Turing machine whose size of all tapes while computing $f(\vec{x})$ is logarithmic in length of $\vec{x}$ and values in $\vec{x}$, assuming binary representation, provided that additional aggregate functions in $\mathcal{F}$ are invariant under permutation of arguments.

Terms. In order to specify values for auxiliary labellings we use *terms*. A term $t(\vec{x})$ is defined by the following BNF:

$$t(\vec{x}) ::= c \mid \lambda(\vec{y}) \mid [Q(\vec{y})] \mid \min_{\lambda,\pi} Q(\vec{y},\pi) \mid \max_{\lambda,\pi} Q(\vec{y},\pi)$$
$$\mid y = y \mid f(t(\vec{y}),\ldots,t(\vec{y})) \mid f'(\{t(x)\colon t(x,\vec{y})\})$$

where $\vec{x}$ is a vector of node variables, x is a fresh node variable, c is a constant, λ is a labelling, Q is a PRA query in G, $f \in \mathcal{F}$, $f' \in \mathcal{F}$ is aggregate, $\vec{y}$ ranges over vectors of variables among $\vec{x}$ and y ranges over variables among $\vec{x}$.

Let G be a graph. A *variable instantiation* $\eta^G : \vec{x} \to V$ in G is a function that maps variables in $\vec{x}$ to nodes of G. Such a function extends canonically to subvectors of $\vec{x}$. Below we inductively extend variable instantiations to terms. If G is clear from the context, we write $t(\vec{v})$ as a shorthand of $\eta^G(t(\vec{x}))$, where $\eta^G(\vec{x}[i]) = \vec{v}[i]$ for all i.

1. $\eta^G(c) = c$, where c is a constant,
2. $\eta^G(\lambda(\vec{y})) = \lambda(\eta^G(\vec{y}))$, where λ is a labelling of G
3. $\eta^G([Q(\vec{y})])$ is 1 if $Q(\eta^G(\vec{y}))$ holds in G and 0 otherwise,
4. $\eta^G(\min_{\lambda,\pi} Q(\vec{y},\pi))$ is the minimum of values of $\lambda[p]$, defined as in the arithmetical constraints, over all paths p such that $Q(\eta^G(\vec{y}),p)$ holds in G,
5. $\eta^G(\max_{\lambda,\pi} Q(\vec{y},\pi)) = \max(\{\lambda[p] \mid Q(\eta^G(\vec{y}),p)\})$,
6. $\eta^G(y = y')$ is 1 if $\eta^G(y) = \eta^G(y')$ and 0 otherwise,
7. $\eta^G(f(t_1(\vec{y}_1),\ldots,t_k(\vec{y}_k))=f(\eta^G(t_1(\vec{y}_1)),\ldots,\eta^G(t_k(\vec{y}_k)))$,
8. $\eta^G(f(\{t(x){:}t'(x,\vec{y})\})) = f(t(v_1),\ldots,t(v_n))$, where $v_1,\ldots,v_n$ are all nodes v of G satisfying $t'(v,\eta^G(\vec{y})) = 1$.

Auxiliary labellings. Consider a term $t(\vec{x})$ and a graph G, which does not have a labelling λ. We define the graph $G[\lambda{:=}t]$ as the graph G extended with the labelling λ such that $\lambda(\vec{v}) = t(\vec{v})$ for all $\vec{v} \in V^k$. We call λ an *auxiliary labelling* of G. We write $G[\lambda_1{:=}t_1,\ldots,\lambda_n{:=}t_n]$ to denote the results of successively adding labellings $\lambda_1,\ldots,\lambda_n$ to the graph G, i.e., $G[\lambda_1{:=}t_1][\lambda_2{:=}t_2]\ldots[\lambda_n{:=}t_n]$.

Language OPRA. An OPRA query is an expression of the form `LET` O `IN` Q', where Q' is a PRA query, O is of the form $\lambda_1{:=}t_1,\ldots,\lambda_n{:=}t_n$ and $t_1,\ldots,t_n$ are terms. The query Q holds over graph G, nodes $\vec{v}$ and paths $\vec{p}$, denoted as $Q(\vec{v},\vec{p})$, if and only if $Q'(\vec{v},\vec{p})$ holds over $G[O]$. Note that $Q'(\vec{x})$ can refer to auxiliary labellings $\lambda_1,\ldots,\lambda_n$. The size of Q is the sum of binary representations of terms $t_1,\ldots,t_n$ and the size of query Q'.

3 Examples

We focus on the following scenario: a graph database that corresponds to a map of some area. Each graph's node represents either a *place* or a *link* from one place to another. The graph has four unary labellings and one binary labelling. The labelling λ_{type} represents the type of a place for places (e.g., square, park, pharmacy) or the mode of transport for links (e.g., walk, tram, train); we assume each type is represented by a constant, e.g., c_{square}, c_{park}. The labelling λ_{attr} represents attractiveness (which may be negative, e.g., in unsafe areas), and λ_{time} represents time. The binary labelling λ_{E} represents edges: for nodes v_1, v_2, the value $\lambda_{\text{E}}(v_1,v_2)$ is 1 if there is an edge from v_1 to v_2 and 0 otherwise. For example, the graph on Fig. 2 represents a map with two places: S is a square and P is a park. There are three nodes representing links: node W represents moving from S to P by walking, T moving from S to P by tram and B moving from P to S by bus.

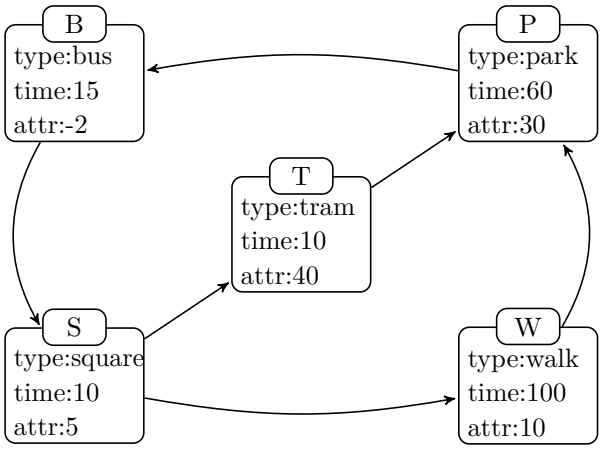


Figure 2 An example of a map-representing graph.

3.1 Language PRA

We begin with the query $Q_{\text{route}}(s,t,\pi)$ stating that there is a path π from a node s to a node t such that each pair of consecutive nodes on this path is connected by an edge given by the edge labelling λ_{E}. Recall that our path constraints of the form $s \to^{\pi} t$ require only that π is a sequence of nodes that starts at s and ends at t. It does not depend on any labelling, in particular λ_{E}. We introduce a regular constraint $\text{route}(\pi)$ defined as $\langle \lambda_{\text{E}}(@_1, @_1') = 1\rangle^* \langle \top \rangle(\pi)$ that states that any two consecutive nodes on π satisfy λ_{E}. As the last node has no successor (i.e., $@_1' = \square$ for the last node), the constraint ends with $\langle \top \rangle$ that is always satisfied. Then, we can express $Q_{\text{route}}(s,t,\pi)$ as

`MATCH NODES` (s,t) `SUCH THAT` $s \to^{\pi} t$ `WHERE` $\text{route}(\pi)$

Sums. The language PRA can express properties of paths' sums. For example, the query below holds iff there is a route from s to t that takes at most 6 hours and its attractiveness is over 100.

`MATCH NODES` (s,t) `SUCH THAT` $s \to^{\pi} t$ `WHERE` $\text{route}(\pi)$
`HAVING` $\lambda_{\text{time}}[\pi] \leq 360 \wedge \lambda_{\text{attr}}[\pi] > 100$

Furthermore, we can compute averages, to some extent. For example, the following arithmetical constraint says that for some path π the average attractiveness of π is at least 4 attractiveness points per minute: $\lambda_{\text{attr}}[\pi] \geq 4\lambda_{\text{time}}[\pi]$.

Multiple paths. We define a query that asks whether there is a route from s to t, such that from every place we can take a tram (e.g., if it starts to rain). We express that by stipulating a route π from s to t and a sequence ρ of tram links, such that every node of π representing a place is connected with the corresponding tram link in ρ. In a way, ρ works as an existential quantifier for nodes of π.

`MATCH NODES` (s,t) `SUCH THAT` $s \to^{\pi} t$
`WHERE` $\text{route}(\pi) \wedge \langle \lambda_{\text{type}}(@_1) = c_{\text{tram}} \rangle^*(\rho) \wedge \text{Link}(\pi,\rho)$

where $\text{Link} = (\langle \lambda_{\text{type}}(@_1) = c_{\text{bus}} \rangle + \langle \lambda_{\text{type}}(@_1) = c_{\text{walk}} \rangle + \langle \lambda_{\text{type}}(@_1) = c_{\text{tram}} \rangle + \langle \lambda_{\text{E}}(@_1, @_2) = 1 \rangle)^*$ states that every node of the first path either is not a place, i.e, it represents any of possible links (by a bus, a walk or a tram), or is connected with the corresponding node of the second path. Note also that in the regular constraint $\langle \lambda_{\text{type}}(@_1) = c_{\text{tram}} \rangle^*(\rho)$ the variable $@_1$ represents the current node of the path ρ, whereas, in $\text{Link}(\pi,\rho)$ the variable $@_1$ represents the current node of π, and $@_2$ represents the current node of ρ.

3.2 Language OPRA

We show how to employ auxiliary labellings in our queries. For readability, we introduce some syntactic sugar – constructions which do not change the expressive power of OPRA, but allow queries to be expressed more clearly. We use the function symbols $=, \neq$ and Boolean connectives, which can be derived from $\leq$ and arithmetical operations. Also, we use terms $t(x, y)$ in arithmetical constraints, which can be expressed by first defining the labelling $\lambda_{\mathrm{t}}(x, y) := t(x, y)$, defining additional paths $\rho_1 = x, \rho_2 = y$ of length 1, and using $\lambda_{\mathrm{t}}[\rho_1, \rho_2]$.

Processed labellings. Online route planners often allow to look for routes which do not require much walking. The following query asks whether there exists a route from s to t such that the total walking time is at most 10 minutes. To express it, we define a labelling $\lambda_{\mathrm{t\_walk}}(x)$, which is the time of x for x that are walking links, and 0 otherwise.
LET $\lambda_{\mathrm{t\_walk}}(x) := (\lambda_{\mathrm{type}}(x) = c_{\mathrm{walk}}) \cdot \lambda_{\mathrm{time}}(x)$ IN
MATCH NODES (s, t) SUCH THAT $s \to^{\pi} t$
WHERE route(π) HAVING $\lambda_{\mathrm{t\_walk}}[\pi] \leq 10$

Nested queries. It is often advisable to avoid crowded places, which are usually the most attractive places. We write a query that holds for routes that are always at least 10 minutes away from any node with attractiveness greater than 100. We define a labelling $\lambda_{\mathrm{crowded}}(x)$ as
[MATCH NODES (x) SUCH THAT $x \to^{\pi} y$ WHERE
route$(\pi) \wedge \langle\top\rangle^* \langle \lambda_{\mathrm{attr}}(@_1) > 100 \rangle(\pi)$ HAVING $\lambda_{\mathrm{time}}[\pi] \leq 10$]
Notice that π and y are existentially quantified. We check whether the value of $\lambda_{\mathrm{crowded}}$ is 0 for each node of the path π.
MATCH PATHS (π) WHERE route$(\pi) \wedge \langle \lambda_{\mathrm{crowded}}(@_1) = 0 \rangle^*(\pi)$

Nodes' neighbourhood. "Just follow the tourists" is an advice given quite often. With OPRA, we can verify whether it is a good advice in a given scenario. A route is called *greedy* if at every position, the following node on the path is the most attractive successor. We define a labelling $\lambda_{\mathrm{MAS}}(x, y)$ that is 1 if y is the most attractive successor of x, and 0 otherwise: $\lambda_{\mathrm{E}}(x, y) \wedge (\textsc{Count}(\{\lambda_{\mathrm{attr}}(z) \colon \lambda_{\mathrm{E}}(x, z) \wedge \lambda_{\mathrm{attr}}(z) \geq \lambda_{\mathrm{attr}}(y)\}) = 1)$. We express that there is a greedy route from s and t.
MATCH NODES (s, t) SUCH THAT $s \to^{\pi} t$
WHERE $\langle \lambda_{\mathrm{MAS}}(@_1, @_1') = 1 \rangle^* \langle\top\rangle(\pi)$

Properties of paths' lengths. In route planning, we often have to balance time, money, attractions, etc. The following query asks whether is it possible to get from s to t in a shortest time possible, in the same time maximising the attractiveness of the route. Recall that Q_{route} is a PRA query defined in Subsection 3.1.
MATCH NODES (s, t) SUCH THAT $s \to^{\pi} t$ WHERE route(π)
HAVING $(\lambda_{\mathrm{attr}}[\pi] = \max_{\lambda_{\mathrm{attr}}, \rho} Q_{\mathrm{route}}(s, t, \rho)) \wedge$
$(\lambda_{\mathrm{time}}[\pi] = \min_{\lambda_{\mathrm{time}}, \rho} Q_{\mathrm{route}}(s, t, \rho))$

Registers. Registers are an important concept often in graph query languages. For instance, to express that two paths have a non-empty intersection, we load a (non-deterministically picked) node from the first path to a register and check whether it occurs in the second path. The following query asks whether there exists a route from a club s to a club t on which the attractiveness of visited clubs never decreases. In the register-based approach, we achieve this by storing the most recently visited club in a separate register. Here, we

express this register using an additional path ρ, storing the values of the register, and a labelling $\lambda_{\mathrm{r}}(x', y, y')$ which states that $y' = x'$ if x' is a club, and $y' = y$ otherwise, defined as $(\lambda_{\mathrm{type}}(x') = c_{\mathrm{club}} \Rightarrow y' = x') \wedge (\lambda_{\mathrm{type}}(x') \neq c_{\mathrm{club}} \Rightarrow y = y')$.
`MATCH NODES` (s,t) `SUCH THAT` $s \to^{\pi} t \wedge s \to^{\rho} t$
`WHERE` $\mathrm{route}(\pi) \wedge \mathrm{ends}(\pi) \wedge \mathrm{regs}(\pi,\rho) \wedge \mathrm{inc}(\rho)$
where $\mathrm{ends} = \langle \lambda_{\mathrm{type}}(@_1) = c_{\mathrm{club}} \rangle \langle \top \rangle^* \langle \lambda_{\mathrm{type}}(@_1) = c_{\mathrm{club}} \rangle$ states that the both ends of a path are clubs, $\mathrm{regs} = \langle \lambda_{\mathrm{r}}(@'_1, @_2, @'_2) = 1 \rangle^* \langle \top \rangle$ ensures that at each position the second path contains the most recently visited club along the first path, and $\mathrm{inc} = \langle \lambda_{\mathrm{attr}}(@_1) \leq \lambda_{\mathrm{attr}}(@'_1) \rangle^* \langle \top \rangle$ checks that the attractiveness never decreases.

4 Expressive power

We compare the expressive power of OPRA and other query languages for graph databases from the literature. We prove the results depicted in Figure 1: that OPRA subsumes ECRPQ, ECRPQ with linear constraints [7] and LARE [23] query languages.

These query languages assume a different notion of graphs from the one considered in this paper. We call graphs as defined in these papers *data graphs*. A *data graph* is a tuple $G = \langle V, E, \lambda \rangle$ where V is a finite set of nodes, $E \subseteq V \times \Sigma \times V$ is a set of edges labelled by a finite alphabet Σ, and $\lambda : V \to \mathbb{Z}^K$ is a labelling of nodes by vectors of K integers. A path in G is a sequence of interleaved nodes and edge labels $v_0 e_1 v_1 \dots v_k$ such that for every $i < k$ we have $E(v_i, e_{i+1}, v_{i+1})$.

The difference between graphs and data graphs is mostly syntactical, yet it prevents us from comparing directly the languages of interest. To overcome this problem, we define the *standard embedding*, which is a natural transformation of data graphs to graphs. For a data graph $G = \langle V, E, \lambda \rangle$ with edges labelled by Σ and nodes labelled by $\mathbb{Z}^K$, we define the graph $G^E = (V^E, \lambda_1^E, \dots, \lambda_K^E, \lambda_{K+1}^E)$, called the standard embedding of G, such that (1) $V^E = V \cup \Sigma$, (2) for every $i \in \{1, \dots, K\}$ and every $v \in V$ we have $\lambda_i^E(v)$ equal to the i-th component of $\lambda(v)$, (3) for every $i \in \{1, \dots, K\}$ and every $v \in \Sigma$ we have $\lambda_i^E(v) = 0$, and (4) for all $v_1, v_2, v_3 \in V^E$ we have $\lambda_{K+1}^E(v_1, v_2, v_3) = 1$ if $(v_1, v_2, v_3) \in E$ and $\lambda_{K+1}^E(v_1, v_2, v_3) = 0$ otherwise. Observe that every node v (resp., every path p) in G corresponds to the unique node v^E (resp., path p^E) in G^E.

A query Q_1 on data graphs is equivalent w.r.t. the standard embedding, *se-equivalent* for short, to a query Q_2 on graphs if for all data graphs G, nodes $\vec{v}$ in G and paths $\vec{p}$ query $Q_1(\vec{v}, \vec{p})$ holds in G if and only if $Q_2(\vec{v}^E, \vec{p}^E)$ holds in G^E, where $G^E, \vec{v}^E, \vec{p}^E$ result from the standard embedding of respectively $G, \vec{v}, \vec{p}$. We say that OPRA *subsumes* a query language $\mathcal{L}$ if every query in $\mathcal{L}$ can be transformed in polynomial time to an se-equivalent OPRA query.

LARE. Queries in LARE are built from arithmetical regular expressions, which extend regular expressions with registers storing nodes and arithmetical operations on labels of the nodes stored in registers (which are natural numbers). We briefly discuss how to express the three main building blocks of LARE expressions: *edge constraints* specifying labels of edges, *register constraints* specifying values of registers, and *register assignments* specifying how registers change.

OPRA queries over the standard embedding can specify labels of edges in the original graph and hence can express edge constraints. Next, we arithmetize all logical operations assuming *true*:= 1 and *false*:= 0. With that, we can show by structural induction that OPRA labellings can express register constraints (e.g., construction $C \vee C'$ can be expressed by term $\max(t_C, t_{C'})$). Finally, we can express registers with additional paths as discussed in Section 3.2.

OPRA language is stronger than LARE. The set of (vectors of) paths satisfying a given LARE query is regular. Therefore, for a fixed LARE query $Q(\pi_1, \pi_2)$, we can decide whether $\forall\pi_1\exists\pi_2 Q(\pi_1, \pi_2)$ holds in a given graph G in polynomial space in G. The set of paths satisfying an OPRA query Q is also related to automata, but due to linear constraints we can express properties of weighted automata. We can define an OPRA query $Q^U(\pi_1, \pi_2)$ which interprets the input graph G as a weighted automaton, path π_1 as an input word and π_2 as a run r on π_1; query Q^U holds only if the value of r is at most 0. Then, $\forall\pi_1\exists\pi_2 Q(\pi_1, \pi_2)$ holds only if the value of every word (w.r.t. the weighted automaton corresponding to G) is at most 0. Such a problem for weighted automata is called (quantitative) universality problem and it is undecidable [1]. Therefore, checking whether a given graph G satisfies $\forall\pi_1\exists\pi_2 Q(\pi_1, \pi_2)$ is undecidable. Thus, no LARE query is se-equivalent to Q^U.

▶ **Theorem 1.** *(1) OPRA subsumes LARE. (2) There is an OPRA query Q with no LARE query Q' se-equivalent to Q.*

ECRPQ with linear constraints. ECRPQ has been extended with linear constrains (ECRPQ+LC) [7], expressing that a given vectors paths $\vec{\pi}$ satisfying a given ECRPQ query satisfies linear inequalities, which specify the multiplicity of edge labels in various components of $\vec{\pi}$. Language OPRA subsumes LARE, which extends ECRPQ, and hence OPRA subsumes ECRPQ. Linear constraints can be expressed by arithmetical constraints of OPRA and hence OPRA subsumes ECRPQ+LC. Moreover, linear constrains are unaffected by nodes' labels and hence ECRPQ+LC cannot express a PRA query saying "the sum of integer labels of nodes along path p is positive". Thus, we have the following.

▶ **Theorem 2.** *(1) OPRA subsumes ECRPQ+LC. (2) There is an OPRA query Q with no ECRPQ+LC query Q' se-equivalent to Q.*

5 The query answering problem

The query-answering problem asks, given an OPRA $Q(\vec{x}, \vec{\pi})$, a graph G, nodes $\vec{v}$ and paths $\vec{p}$ of G, whether $Q(\vec{v}, \vec{p})$ holds in G. We are interested in the *data complexity* of the problem, where the size of a query is treated as constant, and *combined complexity*, where there is no such restriction.

To obtain the desired complexity results, we assume that the absolute values of the graph labels are polynomially bounded in the size of a graph. This allows us to compute arithmetical relations on these labels in logarithmic space. Without such a restriction, the data complexity of the query-answering problem we study is NP-hard by a straightforward reduction from the knapsack problem.

We state the complexity bounds as follows.

▶ **Theorem 3.** *The query answering problem for OPRA queries with bounded number of auxiliary labellings is* PSpace*-complete and its data complexity is* NL*-complete.*

The emptiness problem (whether there exist nodes $\vec{v}$ and paths $\vec{p}$ such that a given OPRA query Q holds for $\vec{v}, \vec{p}$ in a given graph G) has the same complexity; this follows from the fact that a query $Q(\vec{x}, \vec{\pi})$ is non-empty in G iff $Q(\epsilon, \epsilon)$ (same query without free variables) holds in G.

The lower bounds in Theorem 3 follow from the PSpace-hardness of ECRPQ [7], as discussed in Section 4, and for the NL-hardness of the reachability problem.

Recall that an OPRA query is of the form `LET` O `IN` Q', where Q' is a PRA query, O is of the form $\lambda_1{:=}t_1, \ldots, \lambda_n{:=}t_n$ and $t_1, \ldots, t_n$ are terms. Also, by $|O|$ we denote the number

of labellings defined in O. The upper bound in Theorem 3 follows directly from the following lemma.

▶ **Lemma 4.** *For every fixed $s \geq 0$, we have:*

(1) *Given a graph G and an OPRA query $Q :=$ `LET` O `IN` Q' such that $|O| \leq s$, we can decide whether Q holds in G in non-deterministic polynomial space in Q and non-deterministic logarithmic space in G.*

(2) *Given a graph G and an OPRA query $Q :=$ `LET` O `IN` Q' such that $|O| \leq s$, we can compute $\min_{\lambda,\pi} Q(\vec{y}, \pi)$ (resp., $\max_{\lambda,\pi} Q(\vec{y}, \pi)$) non-deterministically in polynomial space in Q and logarithmic space in G. The computed value is either polynomial in G and exponential in Q, or $-\infty$ (resp., ∞).*

We first prove the upper bounds for PRA (i.e., for $s = 0$), and then extend the results to OPRA.

5.1 Language PRA

Assume a PRA query $Q =$ `MATCH NODES` $\vec{x}$`, PATHS` $\vec{\pi}$ `SUCH THAT` P `WHERE` R `HAVING` A. We prove the results in two steps. First, we construct a Turing machine of a special kind (later on called QAM) that represents graphs, called *answer graphs*, with distinguished initial and final nodes, such that every path from an initial node to a final node in this graph is an encoding of paths that satisfy constraints P and R of Q in graph G (for some instantiation of variables $\vec{x}$). These graphs are augmented with the computed values of expressions that appear in arithmetical constraints A. Then, we prove that checking whether in an answer graph there is a path from an initial node to a final node that encodes a path in G satisfying A can be done within desired complexity bounds. Deriving values for $\vec{x}$ from computed paths is straightforward.

The first step is an adaptation of the technique commonly used in the field, e.g., in [7, 23]. We encode vectors $(p_1, \ldots, p_n)$ of paths of nodes from some V as a single path $p_1 \otimes \ldots \otimes p_n$ over the product alphabet V^n (shorter paths are padded with $\square$).

Answer graphs. Consider a graph G with nodes V, its paths $\vec{p}$ and an OPRA query $Q =$ `MATCH NODES` $\vec{x}$`, PATHS` $\vec{\pi}$ `SUCH THAT` P `WHERE` R `HAVING` $\bigwedge_{i=1}^{m} A_i \leq c_i$ with $|\vec{p}| = |\vec{\pi}|$ and existentially quantified path variables $\vec{\pi}'$. Let $k = |\vec{\pi}| + |\vec{\pi}'|$. The *answer graph* for Q on G, $\vec{p}$ is a triple (G', S, T), where $S, T \subseteq M_Q \times V^k$;

- G' is a graph with nodes $M_Q \times V^k$, where M_Q is a finite set computed from Q;
- for each $i \leq m$ and a node $(s, v_1, \ldots, v_k) \in M_Q \times V^k$, the labelling $\lambda_i((s, v_1, \ldots, v_k))$ is defined as the value of the arithmetic constraint A_i over single-node paths v_1, $\ldots$, v_k; and
- a path $q = q_1 \otimes \cdots \otimes q_k$ from (a node in) S to (a node in) T is such that $\lambda_E(v, v') = 1$ for all consecutive v, v' of q iff the paths $q_1, \ldots, q_k$ satisfy $P \wedge R$ and $(q_1, \ldots, q_{|\pi|}) = \vec{p}$.

Intuitively, the labelling λ_E can be defined in such a way that along paths of G' the V^k-components of the nodes correctly encode paths of Q satisfying the path constraints P and the M_Q components store valid runs of automata corresponding to the regular constraints R.

QAMs. Answer graphs can be represented (on-the-fly) in logarithmic space. A Query Applying Machine (QAM) is a non-deterministic Turing Machine which works in logarithmic space and only accepts inputs encoding tuples of the form (G, t, w), where G is a graph and t is a symbol among V, λ, S, T.

For a graph G and $k \geq 0$, a QAM M *gives* a graph $G_k^M = (V, \lambda_1, \ldots, \lambda_k)$ and sets of nodes S_G^M, T_G^M such that:

- V consists of all the *nodes* v s.t. M accepts on (G, V, v),
- λ_i is such that $\lambda_i(\vec{v}) = n$ iff M accepts on $(G, \lambda, (i, \vec{v}, n))$
- S_G^M (resp., T_G^M) consists of $v \in V$ such that M accepts on (G, S, v) (resp., (G, T, v)).

For soundness, we require that for each G, i and $\vec{v}$ there is exactly one n such that M accepts on $(G, \lambda, (i, \vec{v}, n))$.

▶ **Lemma 5.** *For a given query Q and paths $\vec{p}$, we can construct in polynomial time a QAM M^Q such that for every graph G, machine M^Q gives an answer graph for Q on $G, \vec{p}$.*

The second step amounts to the following lemma.

▶ **Lemma 6.** *Let G be a graph and M^Q a QAM, such that M^Q gives the graph $G_m^{M^Q}$ and the sets $S_G^{M^Q}$ and $T_G^{M^Q}$. Let Π be the set of paths from $S_G^{M^Q}$ to $T_G^{M^Q}$ satisfying $\bigwedge_{i=1}^{m} \lambda_i[\pi] \leq c_i$ in $G_m^{M^Q}$. Checking emptiness of Π can be done non-deterministically in polynomial space in Q and logarithmic space in G.*

To solve the emptiness problem from Lemma 6, we need to solve the weighted reachability problem for the corresponding answer graph, which is a graph labelled by vectors of integers. This problem is equivalent to the $\mathbb{Z}$-reachability problem for vector addition systems with states (VASS) [9]. The latter problem has polynomial size solutions. The answer graph is exponential in Q and polynomial in G. Therefore, if Π from Lemma 6 is non-empty, then it contains a path of exponential size in Q and polynomial size in G, and hence its existence can be verified non-deterministically in polynomial space in Q and logarithmic space in G.

5.2 Language OPRA

Assume $O = \lambda_1{:=}t_1, \ldots, \lambda_s{:=}t_s$. We show by induction on s that the values of the labellings of a graph $G[O]$ can be non-deterministically computed in space polynomial in O.

▶ **Lemma 7.** *Let s be fixed. For a graph G and $O = \lambda_1{:=}t_1, \ldots, \lambda_s{:=}t_s$, the value of each labelling of $G[O]$ can be non-deterministically computed in polynomial space in O and logarithmic space in G.*

The proof studies all the possible constructors of terms. In the case of subqueries, we apply the inductive assumption, i.e., Theorem 3 for a query with fewer auxiliary labellings. The case of minimum follow from a counterpart of Lemma 6 for the problem of computing $\min_{p \in \Pi} \lambda_i[p]$, which can be proved by deducing from [8] that $\min_{p \in \Pi} \lambda_i[p]$ is either $-\infty$, $+\infty$ or exponential in Q and polynomial in G. Therefore, it can be computed using Lemma 6 and the bisection method. The case for the maximum is symmetric. Finally, application of a function symbol to terms or ranges can be implemented in the expected complexity.

Now we give some intuition for the proof of Lemma 4. To solve the query answering problem for OPRA, for a given query `LET` O `IN` Q' and a graph G, we first build a QAM $M^{Q'}$, as in Lemma 5. $M^{Q'}$ may refer to labellings from O, not defined in G. We change it so that whenever it wants to access a value of one of labellings defined in O, it instead runs a procedure guaranteed by Lemma 7. Finally, we use Lemma 6 to determine the result.

6 Conclusions

We defined a new query language for graph databases, OPRA and demonstrated its expressive power in two ways. We presented examples of natural properties and OPRA queries expressing

them in an organized, modular way. We showed that OPRA strictly subsumes query languages ECRPQ+LC and LARE. Despite additional expression power, the complexity of the query-answering problem for OPRA matches the complexity for ECRPQ+LC and LARE.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *Automated Technology for Verification and Analysis*, pages 482–491. Springer, 2011.

2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of Modern Graph Query Languages. *CoRR, abs/1610.06264*, 2016. `arXiv:1610.06264`.

3 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *PODS 2014*, pages 14–26. ACM, 2014.

4 Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. DL-lite in the light of first-order logic. In *Proceedings of the national conference on artificial intelligence*, volume 22, page 361. AAAI Press; MIT Press, 2007.

5 Pablo Barceló. Querying graph databases. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS13)*, pages 175–188, 2013.

6 Pablo Barceló, Gaëlle Fontaine, and Anthony W. Lin. Expressive path queries on graph with data. *Logical Methods in Comp. Sci.*, 11(4), 2015.

7 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37:31:1–31:46, 2012.

8 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSpace-complete. *CoRR*, abs/1412.4259, 2014.

9 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 32–43. IEEE Computer Society, 2015. `doi:10.1109/LICS.2015.14`.

10 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.

11 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*, volume 6, pages 260–270, 2006.

12 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proceedings of the 7th International Conference Principles of Knowledge Representation and Reasoning (KR00)*, pages 176–185, 2000.

13 Mariano Consens and Alberto Mendelzon. Low complexity aggregation in GraphLog and Datalog. *Theor. Comp. Sci.*, 116(1):95–116, 1993.

14 Mariano P. Consens and Alberto O. Mendelzon. Graphlog: a visual formalism for real life recursion. In Daniel J. Rosenkrantz and Yehoshua Sagiv, editors, *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, pages 404–416. ACM Press, 1990. `doi:10.1145/298514.298591`.

15 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *Proc. of the ACM Special Interest Group on Management of Data (SIGMOD87)*, pages 323–330, 1987.

16 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. G+: recursive queries without recursion. In *Expert Database Conf.*, pages 645–666, 1988.

17 Richard Cyganiak, Markus Lanthaler, and David Wood. RDF 1.1 concepts and abstract syntax. W3C Recommendation, 2014. URL: `http://www.w3.org/TR/2014/REC-rdf11-concepts/`.

18 Stéphane Demri, Ranko Lazic, and David Nowak. On the freeze quantifier in Constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

19 C.C. Elgot and J.E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965.

20 Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *Proceedings of the 30th Annual Symposium on Logic in Computer Science (LICS15)*, pages 329–340, 2015.

21 Diego Figueira and Leonid Libkin. Synchronizing relations on words. *Theory Comput. Syst.*, 57(2):287–318, 2015.

22 Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993. `doi:10.1016/0304-3975(93)90230-Q`.

23 Michał Graboń, Jakub Michaliszyn, Jan Otop, and Piotr Wieczorek. Querying data graphs with arithmetical regular expressions. In *Proceedings of the 25rd International Joint Conference on Artificial Intelligence (IJCAI16)*. AAAI Press, 2016. URL: `http://www.ii.uni.wroc.pl/~jmi/papers/ijcai16.pdf`.

24 Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, 2013. URL: `https://www.w3.org/TR/sparql11-query/`.

25 Jelle Hellings, Bart Kuijpers, Jan Van den Bussche, and Xiaowang Zhang. Walk logic as a framework for path query languages on graph databases. In *Proceedings of the 16th International Conference on Database Theory (ICDT13)*, pages 117–128, 2013.

26 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

27 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003*, pages 681–696, 2003.

28 Anthony C. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.

29 Eryk Kopczyński and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proceedings of the 25th Symposium on Logic in Comp. Sci. (LICS10)*, pages 80–89, 2010.

30 Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, 2016.

31 Leonid Libkin, Juan L. Reutter, and Domagoj Vrgoc. Trial for RDF: adapting graph query languages for RDF data. In *PODS 2013*, pages 201–212, 2013.

32 Alberto Mendelzon and Peter Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computation*, 24(6):1235–1258, 1995.

33 Jakub Michaliszyn, Jan Otop, and Piotr Wieczorek. Querying best paths in graph databases. *CoRR*, abs/1710.04419, 2017. URL: `http://arxiv.org/abs/1710.04419`.

34 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

35 Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. URL: `http://www.w3.org/TR/rdf-sparql-query/`.

36 Bruno Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the American Mathematical Society*, 284(1):203–218, 1984.

37 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of the 20th Int. Workshop on Computer Science Logic (CSL06)*, pages 41–57, 2006.

38 The Neo4j Team. The Neo4j Manual v3.1., 2017. URL: `http://neo4j.com/docs/`.

39 Peter T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.

Popular Matchings with Lower Quotas*

Meghana Nasre[1] and Prajakta Nimbhorkar[2]

1 **Indian Institute of Technology, Madras, India**
`meghana@cse.iitm.ac.in`
2 **Chennai Mathematical Institute, India**
`prajakta@cmi.ac.in`

Abstract

We consider the well-studied Hospital Residents (HR) problem in the presence of lower quotas (LQ). The input instance consists of a bipartite graph $G = (\mathcal{R} \cup \mathcal{H}, E)$ where $\mathcal{R}$ and $\mathcal{H}$ denote sets of residents and hospitals, respectively. Every vertex has a preference list that imposes a strict ordering on its neighbors. In addition, each hospital h has an associated upper-quota $q^+(h)$ and a lower-quota $q^-(h)$. A matching M in G is an assignment of residents to hospitals, and M is said to be *feasible* if every resident is assigned to at most one hospital and a hospital h is assigned at least $q^-(h)$ and at most $q^+(h)$ residents.

Stability is a de-facto notion of optimality in a model where both sets of vertices have preferences. A matching is *stable* if no unassigned pair has an incentive to deviate from it. It is well-known that an instance of the HRLQ problem need not admit a feasible stable matching. In this paper, we consider the notion of popularity for the HRLQ problem. A matching M is *popular* if no other matching M' gets more votes than M when vertices vote between M and M'. When there are no lower quotas, there always exists a stable matching and it is known that every stable matching is popular.

We show that in an HRLQ instance, although a feasible stable matching need not exist, there is always a matching that is popular in the set of feasible matchings. We give an efficient algorithm to compute a maximum cardinality matching that is popular amongst all the feasible matchings in an HRLQ instance.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases bipartite matchings, preferences, hospital residents, lower-quota, popular matchings

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.44

1 Introduction

In this paper we consider the Hospital Residents problem in the presence of Lower Quotas (HRLQ). The input to our problem is a bipartite graph $G = (\mathcal{R} \cup \mathcal{H}, E)$ where $\mathcal{R}$ denotes the set of residents, and $\mathcal{H}$ denotes the set of hospitals. Every resident as well as hospital has a non-empty preference ordering over a subset of elements of the other set. Every hospital $h \in \mathcal{H}$ has a non-zero upper-quota $q^+(h)$ denoting the maximum number of residents that can be assigned to h. In addition, every hospital h also has a non-negative lower-quota $q^-(h)$ denoting the minimum number of residents that have to be assigned to h. The goal is to assign residents to hospitals such that the upper and lower quotas of all the hospitals are respected (that is, it is feasible) as well as the assignment is *optimal* with respect to the preferences of the participants.

* A full version of the paper is available at [14], `https://arxiv.org/abs/1704.07546`.

▶ **Definition 1.** A feasible matching M in $G = (\mathcal{R} \cup \mathcal{H}, E)$ is a subset of E such that $|M(r)| \leq 1$ for each $r \in \mathcal{R}$ and $q^-(h) \leq |M(h)| \leq q^+(h)$ for each $h \in \mathcal{H}$, where $M(v)$ is the set of neighbors of v in M.

Stability is a de-facto notion of optimality in settings where both sides have preferences. A matching M (not necessarily feasible) is said to be stable if there is no *blocking pair* with respect to M. A resident-hospital pair (r, h) blocks M if r is unmatched in M or prefers h over $M(r)$, and either $|M(h)| < q^+(h)$ or h prefers r over at least one resident in $M(h)$.

There are simple instances of the HRLQ problem where there is no feasible matching that is stable. We give an example here: Let $\mathcal{R} = \{r\}, \mathcal{H} = \{h_1, h_2\}$, $q^+(h_1) = q^+(h_2) = 1$, $q^-(h_1) = 0$, and $q^-(h_2) = 1$. Let preference list of r be $\langle h_1, h_2 \rangle$. That is, r prefers h_1 over h_2. The only stable matching here is $M_1 = \{(r, h_1)\}$ which is not feasible as $|M_1(h_2)| < q^-(h_2)$. On the other hand, the only feasible matching $M_2 = \{(r, h_2)\}$ is not stable as (r, h_1) is a blocking pair with respect to M_2. This raises the question: given an HRLQ instance G, does G admit a feasible stable matching? This can be answered by constructing an HR instance G^+ by disregarding the lower quotas of all hospitals in G. It is well-known that the Gale-Shapley algorithm [5] computes a stable matching M in G^+. Furthermore, from the "Rural Hospitals Theorem" it is known that, in every stable matching of G^+, each hospital is matched to the same capacity [6, 17]. Thus G admits a stable feasible matching if and only if M is feasible for G.

The HRLQ problem is motivated by practical scenarios like assigning medical interns (residents) to hospitals. While matching residents to hospitals, rural hospitals often face the problem of being understaffed with residents, for example the National Resident Matching Program in the US [3, 16, 17]. In such real-world applications declaring that there is no feasible stable matching is simply not a solution. On the other hand, any feasible matching that disregards the preference lists completely is socially unacceptable. We address this issue by relaxing the requirement of stability by an alternative notion of optimality namely *popularity*. Our output matching M has two desirable criteria – firstly, it is a feasible matching in the instance, assuming one such exists, and hence no hospital remains understaffed. Secondly, the matching respects preferences of the participants, in particular, no majority of participants wishes to deviate to another feasible matching in the instance.

Our contribution: We consider the notion of *popularity* for the HRLQ problem. Popularity is a relaxation of stability and can be interpreted as *overall stability*. We define it formally in Section 2. In this work, we present an efficient algorithm for the following two problems in an HRLQ instance.

1. Computing a maximum cardinality matching popular in the set of feasible matchings. We give an $O(|\mathcal{R}| \cdot (|\mathcal{R}| + |\mathcal{H}| + |E|))$ time algorithm for this problem.
2. Computing a popular matching amongst maximum cardinality feasible matchings. We give an $O(|\mathcal{R}|^2 \cdot (|\mathcal{R}| + |\mathcal{H}| + |E|))$ time algorithm for this problem.

Our algorithms are based on ideas introduced in earlier works on stable marriage (SM) and HR problems[11, 3, 15]. In SM and HR problem, a popular matching is guaranteed to exist because a stable matching always exists and it is also popular. On the other hand, in the HRLQ setting even a stable matching may not exist. Yet, we prove that a feasible matching that is popular amongst all feasible matchings always exists and is efficiently computable. We believe that this is not only surprising but also a useful result in practical scenarios. Moreover, our notion of popularity subsumes the notions proposed in [3] and [15] and is more general than both. In [3], popularity is proved using linear programming, but our proofs for popularity are combinatorial.

Overview of the algorithm: Our algorithms are reductions, that is, given an HRLQ instance G, both our algorithms construct instances G' and G'' of the HR problem such that there is a natural way to map a stable matching in G' (respectively, G'') to a feasible matching in G. Moreover, any stable matching in G' (G'') gets mapped to a maximum cardinality matching that is popular amongst all the feasible matchings in G (respectively, a matching that is popular amongst all maximum cardinality matchings in G).

Organization of the paper: We define the notion of popularity in Section 2. The reduction for computing a maximum cardinality popular matching amongst feasible matchings is given in Section 3 and its correctness is proved in Section 4. Finally, Section 5 gives an overview of the algorithm for computing a feasible matching that is popular amongst maximum cardinality feasible matchings.

Related work: The notion of popularity was first proposed by Gärdenfors [7] in the stable-marriage (SM) setting, where each vertex has capacity 1. The notion of popularity has been well-studied since then [2, 10, 11, 9, 4, 12]. A linear-time algorithm to compute a maximum cardinality popular matching in an HR instance is given in [3] and [15] with different notions of popularity. Furthermore, for the SM and HR problem, it is known that a matching that is *popular amongst the maximum cardinality matchings* exists and can be computed in $O(|E|(|\mathcal{R}| + |\mathcal{H}|))$ time [11, 15]. The reductions in our paper are inspired by the work of [3, 4, 11, 15]. In all these earlier works, the main idea is to execute Gale-Shapley algorithm on the HR instance and then allow unmatched residents to propose with *increased priority* [11] certain number of times. As mentioned in [11], this idea was first proposed by Király [13] in the context of approximation algorithms for the SM problem with ties. The HRLQ problem has been recently considered in [1] and [8] in different settings. Very recently, Yokoi [18] considered the notion of *envy free* matchings for the HRLQ problem. Similar to popularity, envy-freeness is also a relaxation of stability. However, unlike popular matchings, every instance of the HRLQ problem need not admit an envy free matching.

2 Notion of popularity

The notion of popularity uses votes from vertices to compare two matchings. For $r \in \mathcal{R}$, and any matching M in G, if r is unmatched in M then, $M(r) = \perp$. A vertex prefers any of its neighbours over $\perp$. For a vertex $u \in \mathcal{R} \cup \mathcal{H}$, let $x, y \in N(u) \cup \{\perp\}$, where $N(u)$ denotes the neighbours of u in G. We define $vote_u(x, y) = 1$ if u prefers x over y, -1 if u prefers y over x and 0 if $x = y$. Given two matchings M_1 and M_2 in the instance, for a resident $r \in \mathcal{R}$, we define $vote_r(M_1, M_2) = vote_r(M_1(r), M_2(r))$.

Voting for a hospital: A hospital h is assigned $q^+(h)$-many votes to compare M_1 and M_2; this is one vote per position of the hospital. If a position is not filled in a matching, we put a $\perp$ there, so that $|M_1(h)| = |M_2(h)| = q^+(h)$.

In our voting scheme, hospital h is indifferent between M_1 and M_2 as far as its $|M_1(h) \cap M_2(h)|$ positions are concerned. To compare between the two sets $M_1(h) \setminus M_2(h)$ and $M_2(h) \setminus M_1(h)$, a hospital can decide any pairing of the elements of these two sets. We denote this correspondence by $\mathbf{corr}_h$ and call it the *correspondence function of* h. Note that $\mathbf{corr}_h$ is dependent on M_1 and M_2. Under this correspondence, for a resident $r \in M_1(h) \setminus M_2(h)$,

$\mathbf{corr}_h(r, M_1, M_2)$ is the resident in $M_2(h) \setminus M_1(h)$ corresponding to r. We define

$$vote_h(M_1, M_2, \mathbf{corr}_h) = \sum_{r \in M_1(h) \setminus M_2(h)} vote_h(r, \mathbf{corr}_h(r, M_1, M_2))$$

A hospital h prefers M_1 over M_2 under $\mathbf{corr}_h$ if $vote_h(M_1, M_2, \mathbf{corr}_h) > 0$. There are several ways for a hospital to define the $\mathbf{corr}_h$ function. For example, a hospital h may decide to order and compare the two sets in the decreasing order of preferences (as in [15]) or in the *most adversarial order* (as in [3]). That is, the order due to which h gives the least votes to M_1 when comparing it with M_2. We believe that our definition offers flexibility to hospitals to compare residents in $M_1(h) \setminus M_2(h)$ and $M_2(h) \setminus M_1(h)$ according to their custom designed criteria. To compare M_1 and M_2, each hospital h fixes $\mathbf{corr}_h$. The disjoint union of these functions, $\mathbf{corr} = \biguplus_h \mathbf{corr}_h$, called the *correspondence function from M_1 to M_2* is then used to define the collective votes of M_1 compared to M_2

$$\Delta(M_1, M_2, \mathbf{corr}) = \sum_{r \in \mathcal{R}} vote_r(M_1, M_2) + \sum_{h \in \mathcal{H}} vote_h(M_1, M_2, \mathbf{corr}_h)$$

We can now define popularity.

▶ **Definition 2.** A matching M_1 is more popular than M_2 (denoted as $M_1 \succ_{\mathbf{corr}} M_2$) under $\mathbf{corr}$ if $\Delta(M_1, M_2, \mathbf{corr}) > 0$. A matching M_1 is *popular* if there is no matching M_2 such that $M_2 \succ_{\mathbf{corr}} M_1$ for any choice of $\mathbf{corr}$ from M_2 to M_1.

It is important to note that, between two matchings M_1 and M_2, matching M_1 may get more votes than M_2 under one correspondence function, but not under another correspondence function. For M_1 to be popular, we require that any other matching M_2 does not get more votes than M_1 under *any* choice of correspondence function. Surprisingly, such a matching indeed exists, as shown in Sections 3 and 4. We also note that both our algorithms (in fact reductions), do not need as an input the correspondence function $\mathbf{corr}$.

Decomposing $M_1 \oplus M_2$: In the one-to-one setting, $M_1 \oplus M_2$ for any two matchings M_1 and M_2 is a collection of vertex-disjoint paths and cycles. Our setting is many-to-one and hence $M_1 \oplus M_2$ has a more complex structure. Here, we recall a simple algorithm from [15] which, given two matchings M_1 and M_2 and a correspondence function $\mathbf{corr}$ from M_1 to M_2, decomposes the edges of $M_1 \oplus M_2$ into (possibly non-simple) alternating paths and cycles. Consider the graph $\tilde{G} = (\mathcal{R} \cup \mathcal{H}, M_1 \oplus M_2)$, for any two feasible matchings of the HRLQ instance. We note that the degree of every resident in $\tilde{G}$ is at most 2 and the degree of every hospital in $\tilde{G}$ is at most $2 \cdot q^+(h)$. Consider any connected component $\mathcal{C}$ of $\tilde{G}$ and let $e \in M_1$ be any edge in $\mathcal{C}$. We show how to construct a unique maximal M_1-alternating path or cycle ρ containing e: Start with $\rho = \langle e \rangle$. Use the following inductive procedure.

1. Let $r \in \mathcal{R}$ be one end-point of ρ, and let $(r, M_1(r)) \in \rho$. We grow ρ by adding the edge $(r, M_2(r))$. Similarly if $(r, M_2(r)) \in \rho$, add $(r, M_1(r))$ to ρ.
2. Let $h \in \mathcal{H}$ be an end-point of ρ, and let the last edge (r, h) on ρ be in $M_1 \setminus M_2$. We extend ρ by adding $\mathbf{corr}_h(r, M_1, M_2)$ if it is not equal to $\perp$. A similar step is performed if the last edge on ρ is $(r, h) \in M_2 \setminus M_1$.
3. We stop the procedure when we complete a cycle (ensuring that the two adjacent residents of a hospital h are $\mathbf{corr}_h$ for each other according to h), or the path can no longer be extended. Otherwise we go to Step 1 or Step 2 as applicable and repeat.

r_1	:	h_1, h_3, h_4, h_5	$(0,1)$	h_1	:	r_1, r_2, r_3
r_2	:	h_2, h_1, h_3	$(0,1)$	h_2	:	r_2, r_3, r_4
r_3	:	h_2, h_1	$(0,1)$	h_3	:	r_1, r_2
r_4	:	h_2	$(0,1)$	h_4	:	r_1
			$(1,1)$	h_5	:	r_1

Figure 1 Resident and hospital preferences in G. The $(0,1)$ beside h_1 denote the lower and upper quotas of h_1 respectively. Preferences can be read as: r_1 prefers h_1 followed by h_3 and so on.

Labels on edges: While comparing M_1 with M_2 using **corr**, the voting scheme induces a label on edges of M_2 with respect to M_1. Let $(r,h) \in M_2$. The label of (r,h) is (a,b) where $a = vote_r(M_1(r), M_2(r))$ and $b = vote_h(\mathbf{corr}_h(r, M_2, M_1), r)$. Thus $a, b \in \{-1, 1\}$. Here it is important to note that $\mathbf{corr}_h$ is a bijection between $M_1(h) \setminus M_2(h)$ and $M_2(h) \setminus M_1(h)$.

3 Maximum cardinality popular matching

We first give some intuition and an example which illustrates the overall idea of our algorithm. We then present a reduction from HRLQ to HR which simulates the algorithm.

At the high-level, our algorithm has three phases. In Phase-0 we simply execute the hospital-proposing Gale-Shapley algorithm on G by disregarding lower quotas of all hospitals. Let M_0 be a matching obtained. Phase-1 is the "second chance phase". In this phase, all hospitals that are *under-subscribed* in M_0 (a hospital h is under-subscribed in M if $|M(h)| < q^+(h)$), propose to residents in their preference list with increased priority and a new matching M_1 is obtained. The priority is simulated by assigning *levels* to hospitals. Phase-0 is executed with all the hospitals at level 0. The priority of a hospital is increased by increasing its level. A resident prefers a higher level hospital to any lower level hospital *irrespective* of the relative positions of the hospitals in the his preference list.

Finally, Phase-2 is the "feasibility phase". If there are *deficient hospitals* in M_1 (h is deficient in M if $|M(h)| < q^-(h)$), they again apply with an even higher priority. This process is repeated until there is no deficient hospital (we prove that $|\mathcal{R}|$ repetitions are sufficient) and every under-subscribed hospital has got a second chance.

Example: Let $G = (\mathcal{R} \cup \mathcal{H}, E)$ where $\mathcal{R} = \{r_1, \ldots, r_4\}$ and $\mathcal{H} = \{h_1, \ldots, h_5\}$. Figure 1 shows the quotas and the preferences of vertices in G.

The first part of the algorithm is an execution of hospital-proposing Gale-Shapley algorithm, with all hospitals at *level*-0. This results in a stable but infeasible matching $M_0 = \{(r_1, h_1^0), (r_2, h_2^0)\}$. As h_3, h_4, h_5 are under-subscribed in M_0, their level is increased, and h_3^1, h_4^1, h_5^1 propose from the beginning of the respective preference lists. The hospital h_3^1 proposes to r_1 which is accepted. This leaves h_1 under-subscribed. However, note that h_1^0 has not yet exhausted its preference list, so h_1^0 proposes to r_3 and this proposal is accepted. Also note that h_4^1 and h_5^1 do not get matched, as the only resident in their preference list, r_1, prefers h_3^1 over them. This results in a larger but infeasible matching $M_1 = \{(r_1, h_3^1), (r_2, h_2^0), (r_3, h_1^0)\}$.

We further increase the level of the deficient hospital h_5, thus h_5^2 proposes to r_1. This triggers a series of proposals (see Figure 2(b) and Figure 2(c) in Appendix A for details) and we finally obtain the feasible matching $M_2 = \{(r_1, h_5^2), (r_2, h_1^1), (r_3, h_2^0)\}$. As all the

under-subscribed hospitals have exhausted their preference lists at *level*-0 and *level*-1 and there are no deficient hospitals, the algorithm terminates.

Note that all the hospitals are allowed to propose at *level*-0, any under-subscribed hospital is allowed to propose at *level*-1 (all hospitals except h_2, in the example), whereas only deficient hospitals are allowed to propose at *level*-2 and higher (h_5 in the example).

3.1 The reduced graph G'

To simulate the above algorithm on an HRLQ instance G, we convert G to an HR instance $G' = (\mathcal{R}' \cup \mathcal{H}', E')$. The main idea is to have in G' multiple copies of every hospital in G – the first two copies have capacity equal to upper-quota and the rest of the copies have capacity equal to lower-quota. Furthermore, we need a suitably large set of dummy residents to ensure that a matching in G' matches at most upper-quota many non-dummy residents across all copies of a hospital. We describe our reduction – we begin with vertices in G'.

The set $\mathcal{H}'$: For each hospital $h \in \mathcal{H}$ we have ℓ copies $h^0, \ldots, h^{\ell-1}$ of h in $\mathcal{H}'$. Here $\ell = 2 + \sum_{h \in \mathcal{H}} q^-(h)$. We need to define the capacities of all hospitals $h \in \mathcal{H}'$ (recall G' is an HR instance, so we do not have lower quotas for $h \in \mathcal{H}'$). For the upper quota of a hospital, we use the term "capacity" in an HR instance whereas "upper quota" in an HRLQ instance. The hospitals in $\mathcal{H}'$ and their capacities are as described below:

$$\begin{aligned} \mathcal{H}' &= \{h^0, \ldots, h^{\ell-1} \mid h \in \mathcal{H}\} \\ \text{Capacities of } h \in \mathcal{H}'\text{: } q^+(h^s) &= q^+(h), \quad s \in \{0, 1\} \\ q^+(h^s) &= q^-(h), \quad s \in \{2, \ldots, \ell - 1\} \end{aligned}$$

We call hospital $h^s \in \mathcal{H}'$ a *level-s* copy of h. Note that, if $h \in \mathcal{H}$ has zero lower-quota, then $h^2, \ldots, h^{\ell-1}$ have zero capacity in $\mathcal{H}'$. As will be seen later, a resident prefers a hospital at a higher level to *any* hospital at a lower level. The following observation is immediate.

▶ **Observation 1.** *For a hospital $h \in \mathcal{H}$, the sum of capacities of all the copies of h in G' is $q_h = 2 \cdot q^+(h) + (\ell - 2) \cdot q^-(h)$.*

The set $\mathcal{R}'$: The set of residents $\mathcal{R}'$ consists of the set $\mathcal{R}$ along with a set of dummy residents $\mathcal{D}_h$ corresponding to every hospital $h \in \mathcal{H}$. The sets $\mathcal{R}'$ and $\mathcal{D}_h$ are as defined below:

$$\mathcal{R}' = \mathcal{R} \cup \left(\bigcup_{h \in \mathcal{H}} \mathcal{D}_h \right) \text{ where } \qquad \mathcal{D}_h = \bigcup_{s \in \{0, \ldots, \ell-2\}} \mathcal{D}_h^s \qquad \forall h \in \mathcal{H}$$

$$\begin{aligned} \text{Here } \mathcal{D}_h^s &= \{d_{h,1}^s, \ldots, d_{h,q^+(h)}^s\}, \quad s \in \{0, 1\} \\ \text{and } \mathcal{D}_h^s &= \{d_{h,1}^s, \ldots, d_{h,q^-(h)}^s\}, \quad s \in \{2, \ldots, \ell - 2\} \end{aligned}$$

We refer to $\mathcal{D}_h$ as *dummy residents corresponding to h* and $\mathcal{D}_h^s$ as *s-th set of dummy residents corresponding to h*. Note that, for a hospital h, for any level s except the last level, we have dummy residents in $\mathcal{D}_h^s$ exactly equal to the capacity of h^s in G'. For $h \in \mathcal{H}$, if $q^-(h) = 0$, then $\mathcal{D}_h^s = \emptyset$ for each $s \in \{2, \ldots, \ell - 2\}$.

▶ **Observation 2.** *For a hospital $h \in \mathcal{H}$, the total number of dummy residents corresponding h in $\mathcal{R}'$ is $|\mathcal{D}_h| = 2 \cdot q^+(h) + (\ell - 3) \cdot q^-(h)$.*

Preference lists: We denote by $\langle list_r \rangle$ and $\langle list_h \rangle$ the preference lists of r and h in G, respectively. Furthermore, $\langle \mathcal{D}_h^s \rangle$ denotes the strict list consisting of elements of $\mathcal{D}_h^s$ in increasing order of indices. Finally, $\circ$ denotes the concatenation of two lists. We now describe the preferences of hospitals and residents in G'.

Hospitals' preference lists: For a hospital h in $\mathcal{H}$, the preference lists of its copies $h^s \in \mathcal{H}'$ for $s \in \{0, 1, \ldots, \ell - 1\}$ are given by:

$$\begin{aligned}
s = 0 \quad &: \quad \langle list_h \rangle \circ \langle \mathcal{D}_h^0 \rangle \\
s = 1 \quad &: \quad \langle \mathcal{D}_h^0 \rangle \circ \langle list_h \rangle \circ \langle \mathcal{D}_h^1 \rangle \\
s = 2 \quad &: \quad \langle d_{h,k(h)}^1, \ldots, d_{h,q^+(h)}^1 \rangle \circ \langle list_h \rangle \circ \langle \mathcal{D}_h^2 \rangle, \qquad k(h) = q^+(h) - q^-(h) + 1 \\
s \in \{3, 4, \ldots, \ell - 2\} \quad &: \quad \langle \mathcal{D}_h^{s-1} \rangle \circ \langle list_h \rangle \circ \langle \mathcal{D}_h^s \rangle \\
s = \ell - 1 \quad &: \quad \langle \mathcal{D}_h^{(\ell-2)} \rangle \circ \langle list_h \rangle
\end{aligned}$$

The preference list of the *level*-2 copy of h is slightly different. In $\mathcal{D}_h^1$ we have $q^+(h)$ dummy residents but we want only $q^-(h)$ many out of them to be in the preference list of h^2.

Residents' preference lists: For any $s \in \{0, \ldots, \ell - 1\}$, and any $r \in \mathcal{R}$, let $\langle list_r \rangle^s$ denote the list obtained by replacing every hospital h in $\langle list_r \rangle$ by its level-s copy h^s. Then the preference list of r in G' is given by:

$$\text{For } r \in \mathcal{R} \quad : \quad \langle list_r \rangle^{\ell-1} \circ \langle list_r \rangle^{\ell-2} \circ \ldots \circ \langle list_r \rangle^0$$

Thus r prefers any level-s hospital to any level-$(s-1)$ hospital. For two hospitals at the same level s, r prefers h^s over h'^s in G' iff r prefers h over h' in G. We now give the preference list of every dummy resident in $\mathcal{D}_h$. Recall that for any $h \in \mathcal{H}$, $k(h) = q^+(h) - q^-(h)$.

$$\begin{aligned}
\text{For } h \in \mathcal{H}, \quad d_{h,i}^s \in \mathcal{D}_h \quad &: \\
s = 0 \quad &: \quad h^0, h^1 \\
s = 1, \quad i \in \{1, \ldots, k(h)\} \quad &: \quad h^1 \\
s = 1, \quad i \in \{k(h) + 1, \ldots, q^+(h)\} \quad &: \quad h^1, h^2 \\
s \in \{2, \ldots, \ell - 2\} \quad &: \quad h^s, h^{s+1}
\end{aligned}$$

3.2 Properties of the stable matching M' in G'

Having described the reduction from an HRLQ instance G to an HR instance G', we now discuss some useful properties of a stable matching M' in G'. With respect to a stable matching M' in G' we introduce the following definitions.

▶ **Definition 3. Level-s resident:** A non-dummy resident $r \in \mathcal{R}'$ is said to be at *level-s* in M' if r is matched to a *level-s* hospital in M'. Let $\mathcal{R}'_s$ denote the set of *level-s* residents.

▶ **Definition 4. Active hospital:** A hospital h^s is said to be *active* in M' if $M'(h^s)$ contains at least one non-dummy resident. Otherwise, (when all positions of h^s are matched to dummy residents), h^s is said to be *inactive*.

In Lemma 5, we prove some invariants for any stable matching M' in G'. These invariants allow us to define a natural map from M' to a matching M in G, and to show that M is feasible as well as popular among feasible matchings.

▶ **Lemma 5.** *The following hold for any stable matching M' in G':*

1. *For any $h \in \mathcal{H}$, M' matches at most $q^+(h)$ non-dummy residents across all its level copies in G'.*

2. *The matching M' in G' leaves only the level-$(\ell-1)$ copy of any hospital (if it exists) under-subscribed.*
3. *Let $h^s \in \mathcal{H}'$ be active in M'. Then,*
   (a) *$M'(h^{s-1})$ contains at least one resident from the $(s-1)$-th set of dummy residents.*
   (b) *For $0 \leq j \leq s-2$, h^j is inactive in M' and all positions of h^j are matched to the j-th set of dummy residents.*
   (c) *For $s+2 \leq j \leq \ell-1$, h^j is inactive in M' and all positions of h^j are matched to the $(j-1)$-th set of dummy residents.*
4. *For any $h \in \mathcal{H}$, at most two consecutive level copies h^s and h^{s+1} are active in M'.*
5. *A level-s resident r in M' does not have any hospital h in its preference list which is active at level-$(s+2)$ or more in M'.*

Proof.

- *Proof of 1*: We first note that the total capacity of all the copies of h in G' is $q_h = 2 \cdot q^+(h) + (\ell-2) \cdot q^-(h)$. Furthermore, the total number of dummy residents for h is given by $|\mathcal{D}_h| = 2 \cdot q^+(h) + (\ell-3) \cdot q^-(h)$. We now show that at most $q^+(h) - q^-(h)$ dummy residents out of $\mathcal{D}_h$ can remain unmatched in a stable matching M'. Assuming this, it is immediate that the total number of non-dummy residents that can be matched across all copies of h is at most $q_h - \{|\mathcal{D}_h| - (q^+(h) - q^-(h))\} = q^+(h)$.
  We now argue that at most $q^+(h) - q^-(h)$ dummy residents of $\mathcal{D}_h$ can remain unmatched in M'. Consider the set of dummy residents corresponding to a hospital $h \in \mathcal{H}$ i.e. $\bigcup_{s=0}^{\ell-2} \mathcal{D}_h^s$. With the exception of h^2, for any h^s, $\mathcal{D}_h^{(s-1)}$ are the most preferred $q^+(h^s)$ dummy residents of h^s. A dummy resident $d_{h,j}^{s-1}$ which is a top choice for h^s cannot remain unmatched in M', else $(d_{h,j}^{s-1}, h^s)$ blocks M'. Thus, these dummy residents can never remain unmatched in M'. The only dummy residents that are not the first choice of any hospital and hence can remain unmatched are the subset of $\mathcal{D}_h^1$ consisting of the first $q^+(h) - q^-(h)$ dummy residents from $\mathcal{D}_h^1$. This is because, by construction of G', only the last $q^-(h)$ dummy residents from $\mathcal{D}_h^1$ are present in the preference list of h^2 as its top $q^+(h^2)$ top-choices. This establishes that the number of dummy residents of $\mathcal{D}_h$ that can remain unmatched in M' is at most $q^+(h) - q^-(h)$.
- *Proof of 2:* Consider a hospital $h \in \mathcal{H}$. For each copy h^s of h in $\mathcal{H}'$, where $s < \ell - 1$, the s-th set of dummy residents have h^s as their first choice. Further, their number is same as $q^+(h^s)$. Thus h^s can not remain under-subscribed in any stable matching M' of G', otherwise these dummy residents will form a blocking pair with h^s.
- *Proof of 3a:* For the sake of contradiction, assume that h^{s-1} is not matched to any resident from the $(s-1)$-th set of dummy residents and still h^s is matched to a non-dummy resident. As there are exactly $q^+(h^s)$ many dummy residents in the preference list of h^s from the $(s-1)$-th set and each dummy resident from the $(s-1)$-th set has only h^{s-1} and h^s in its preference list, this means that there is a dummy resident d from its $(s-1)$-th set of dummy residents unmatched in M'. But h^s prefers any dummy resident in its $(s-1)$-th set of dummy residents over any non-dummy resident. Thus (d, h^s) forms a blocking pair with respect to M', contradicting the stability of M'.
  Proof of 3b: If h^s is active and h^j is matched to a non-dummy resident r for some $0 \leq j \leq s-2$, then $(r, h^{(s-1)})$ is a blocking pair with respect to M'. This is because, as proved above, $h^{(s-1)}$ must be matched to at least one resident in $\mathcal{D}_h^{(s-1)}$, and $h^{(s-1)}$ prefers any non-dummy resident over any dummy resident in $\mathcal{D}_h^{(s-1)}$.
  Proof of 3c: If h^s is active then h^j can not be active for $s+2 \leq j \leq \ell-1$ else $h^{(j-1)}$ must be matched to a resident from $\mathcal{D}_h^{(j-1)}$ as proved above, and then each non-dummy resident r in $M'(h^s)$ forms a blocking pair with h^j contradicting the stability of M'. But

if h^s is active, then h^j can not be matched to a dummy resident from $\mathcal{D}_h^j$ either, otherwise a resident in $M'(h^s)$ forms a blocking pair with h^{j-1}. The latter is true because any resident in $\langle list_h \rangle$ prefers h^j over h^s for $j > s$ and h^j prefers any resident in $list_h$ to any dummy resident in $\mathcal{D}_h^j$. Hence h^j must be matched to only dummy residents in $\mathcal{D}_h^{(j-1)}$.

- *Proof of 4:* Assume the contrary. Thus let h be a hospital such that there are two levels i and j, $j < i-1$, where h^i and h^j are active in M'. Further, assume that h^i is matched to r_i and h^j be matched to r_j. Then, by part 3a above, h^{i-1} must be matched to at least one resident from the $(i-1)$-th set of dummy resident. But, by the structure of preference lists, h^{i-1} prefers a non-dummy resident, and hence r_j, over any resident in the $(i-1)$-th set of dummy residents. Also, r^j prefers h^{i-1} over h^j since $j < i-1$. Thus (r_j, h^{i-1}) forms a blocking pair in G' w.r.t. M', contradicting the stability of M'.
- *Proof of 5:* Let there be an edge (r, h^t) in G' such that r is a level-s resident and $t \geq s+2$ and h^t is active in M'. Then, by part 3a above, h^{t-1} has at least one resident from its $(t-1)$-th set of dummy residents in $M'(h^{s+1})$. As r has edge to h^t, r also has an edge to h^{t-1} by construction of G'. Also, again by construction of G', r prefers any level-$(t-1)$ hospital over any level-s hospital and h^{t-1} prefers any non-dummy resident in its preference list over any dummy resident in its $(t-1)$-th set of dummy residents. Thus (r, h^{t-1}) forms a blocking pair with respect to M' in G', contradicting its stability.

This completes the proof of all invariants. ◀

4 Proof of popularity

In this section, we show how to use the reduction in the previous section to compute a maximum cardinality matching that is popular amongst all feasible matchings. Thus, amongst all feasible matchings, our algorithm outputs the largest popular matching. We call such a matching a *maximum cardinality popular matching*.

Our algorithm reduces the HRLQ instance G to an HR instance G' as described in Section 3. We then compute a stable matching M' in G'. Finally, to obtain a matching M in G we describe a simple map function. For every $h \in \mathcal{H}$, let $M(h) = \mathcal{R} \cap \left(\bigcup_{s=0}^{\ell-1} M'(h^s)\right)$. Note that $M(h)$ denotes the set of non-dummy residents matched to any copy h^s of h in M'. Thus, a resident r is matched to a hospital h in M if and only if r is matched to a level-s copy of h in M' for some $s \in \{0, \ldots, \ell-1\}$. We say that $M = map(M')$.

Division of $\mathcal{R}$ and $\mathcal{H}$ into subsets: We divide the residents and hospitals in G into subsets depending upon a matching M' in G'. Let R_i be the set of non-dummy residents matched to a *level-i* hospital h^i in M'. We define the same set R_i in G as well. Further, define H_j to be the set of hospitals $h \in H$ such that $\mathcal{R} \cap M'(h^j) \neq \emptyset$, that is, *level-$j$* copy h^j of h is active in M'. Define the unmatched residents to be in R_0. Also, a non-lower-quota hospital h such that $M(h) = \emptyset$ is defined to be in H_1, and a lower-quota hospital h with $M(h) = \emptyset$ is defined to be in $H_{\ell-1}$. The following lemma summarizes the properties of the sets R_i and H_j. See full-version [14] for proof.

▶ **Lemma 6.** *For a stable matching M' in G', let $M = map(M')$. The following hold:*

1. *Each hospital is present in at most two sets H_j, H_{j+1} for some j. We say that $h \in H_j \cap H_{j+1}$.*
2. *If $h \in H_j \cap H_{j+1}$, then there is no edge from h to any $r \in R_i$ where $i \leq j-1$.*
3. *All the non-lower-quota hospitals that are under-subscribed in M are in H_1.*
4. *All the deficient lower-quota hospitals from M are in $H_{\ell-1}$.*

5. *If a non-lower-quota hospital is under-subscribed, it has no edge to any resident in R_0. If a lower-quota hospital is deficient, it does not have an edge to any resident in R_i for $i < \ell - 1$. Similarly an unmatched resident does not have an edge to any hospital in $H_1 \cup \ldots \cup H_{\ell-1}$.*
6. *Let $h \in \mathcal{H}$ be such that $|M(h)| > q^-(h)$. Then $h \notin H_2 \cup \ldots \cup H_{\ell-1}$.*

Throughout the following discussion, assume that M is a matching which is a map of a stable matching M' in G' and N is any feasible matching in G. Additionally, whenever we consider the decomposition of $M \oplus N$ into paths and cycles, we use an arbitrary correspondence function **corr**. Using Theorem 7 we prove that M is feasible in G.

▶ **Theorem 7.** *If G admits a feasible matching, then $M = map(M')$ is feasible for G.*

Proof. Suppose M is not feasible. Thus, there is a deficient lower-quota hospital h in M. Let N be a feasible matching in G. Consider decomposition of $M \oplus N$ into (possibly non-simple) paths and cycles using an arbitrary correspondence function **corr** (recall decomposition from Section 2). As h is deficient in M and not deficient in N, there must be a path ρ in $M \oplus N$ ending in h. Moreover, if the other end of ρ is a hospital h' then $|M(h')| - |N(h')| > 0$. Note that in this case, ρ has even-length and hence ends with a M-edge. The other case is where ρ ends in a resident r and hence ends with a N-edge. We consider the two cases below:

- **ρ ends in a hospital h':** As h is deficient in M, $h \in H_{\ell-1}$ by part 4 of Lemma 6. Also, since $|M(h')| > |N(h')| \geq q^-(h')$, by part 6 of Lemma 6, $h' \in H_0 \cup H_1$. Thus ρ starts at $H_{\ell-1}$ and ends in H_0 or H_1. Let $\rho = \langle h, r_1, h_1, r_2, h_2, \ldots, r_t, h_t, r', h' \rangle$, where $(r_i, h_i) \in M$ and $(r', h') \in M$. We show below that such a path ρ can not exist and hence M must be feasible.

  By part 5 of Lemma 6, h has edges only to residents in $R_{\ell-1}$. Hence $r_1 \in R_{\ell-1}$ and hence $h_1 \in H_{\ell-1}$. By part 2 of Lemma 6, h_1 has no edges to residents in $R_0 \cup \ldots \cup R_{\ell-3}$. Therefore $r_2 \in R_{\ell-1} \cup R_{\ell-2}$ and $h_2 \in H_{\ell-1} \cup H_{\ell-2}$. Thus each $h_i \in \rho$ can not be in H_j, for any $j < \ell - i$. But $h' \in H_0 \cup H_1$ and hence $r' \in R_0 \cup R_1$. Therefore $h_t \notin H_3 \cup \ldots \cup H_{\ell-1}$ by part 2 of Lemma 6, otherwise (h_t, r') edge can not exist in G. In other words, ρ has to contain at least one hospital from each level i, $1 \leq i \leq \ell - 1$. Thus $t \geq \ell - 2$. Moreover, all the hospitals in ρ which are in $H_{\ell-1} \cup \ldots \cup H_2$ are lower-quota hospitals. Thus ρ has at least $t + 1 = \ell - 1$ lower-quota hospitals. Note that this count includes repetitions, as a hospital can appear multiple times in ρ. However, any hospital in $H_2 \cup \ldots \cup H_{\ell-1}$ can not be matched to more than $q^-(h)$ residents in M by part 6 and hence can appear at most $q^-(h)$ times on ρ. But then the sum of lower quotas of all the hospitals is $\ell - 2$, contradicting that ρ has a total of $\ell - 1$ occurrences of lower-quota hospitals. Thus such a path ρ can not exist and M must be feasible.
- **ρ ends in a resident r:** Now consider the case where ρ ends at a resident r. Then the last edge on ρ must be a N-edge and hence r is unmatched in M. Therefore $r \in R_0$. Let $\rho = \langle h, r_1, h_1, r_2, h_2, \ldots, r_t, h_t, r \rangle$ where $(r_i, h_i) \in M$ for $1 \leq i \leq t$ and the remaining edges are in N. Consider the first hospital, say h_j on ρ such that $h_j \in H_2$ and for each $h_i, i < j$, $h_i \in H_3 \cup \ldots \cup H_{\ell-1}$. Such an h_j has to exist by the argument given for the previous case. Moreover, $j \geq \ell - 2$ as ρ has to contain at least one hospital from each level as described in the previous case. Thus the number of occurrences of lower-quota hospitals on ρ exceeds the sum of lower quotas and hence such a ρ can not exist.

This completes the proof of the lemma. ◀

In Lemma 8 and Theorem 9 below, we give crucial properties of the division of $\mathcal{R}$ and $\mathcal{H}$ that will be helpful in proving popularity of the matching M which is a map of a stable matching

M' in G'. For both of them, assume that **corr** is an arbitrary correspondence function from N to M. The proofs appear in the full-version [14].

▶ **Lemma 8.** *Let N be any feasible matching. Let $(r,h) \in M$ and $(r',h) \in N$ such that $r' = \boldsymbol{corr}_h(r, M, N)$. Further let $h \in H_j \cap H_{j+1}$ and $r \in R_{j+1}$. Further, let $r' \in R_j$. Then the label on (r',h) edge is $(-1,-1)$.*

Let $\rho = \langle h_0, r_1, h_1, r_2, h_2, \ldots, h_t, r_{t+1}\rangle$ be a path in $M \oplus N$ where $M = map(M')$ and N is any feasible matching in G. Also, label the edges of $N \setminus M$. See Section 2 for the decomposition and labeling of edges.

▶ **Theorem 9.** *Let $\rho = \langle h_0, r_1, h_1, r_2, h_2, \ldots, h_t, r_{t+1}\rangle$ be a path in $M \oplus N$ as described above. Here $(r_k, h_k) \in M$ for all k and $(h_k, r_{k+1}) \in N$ with $r_{k+1} = \boldsymbol{corr}_{h_k}(r_k, M, N)$. Moreover, let $h_0 \in H_p \cap H_{p+1}$ and $r_{t+1} \in R_q$. Then the number of $(1,1)$ edges in ρ is at most the number of $(-1,-1)$ edges plus $q-p$.*

Proof. We prove this by induction on the number of $(-1,-1)$ edges. Note that, except h_0, all the h_is are matched in M', and hence we can consider them at the same level as their matched residents.

Base case: Let ρ have no $(-1,-1)$ edges. As ρ starts at $h \in H_p \cap H_{p+1}$, r_1 has to be in *level*-$(p+1)$ or above. This is because there is no edge from h to a resident in $R_0 \cup \ldots \cup R_{p-1}$, and if $r_1 \in R_p$ then by Lemma 8 the label on (h_0, r_1) must be $(-1,-1)$. By assumption, there is no $(-1,-1)$ edge in ρ. So $r_1 \in R_j$ for some j, $p+1 \le j \le \ell$. Therefore $h_1 \in H_j$.

Thus the path can only use edges from a hospital at a lower level to a resident at the same or higher level. Further, there is no $(1,1)$ edge in $H_k \times R_k$ for any k; otherwise the same edge blocks M' in G' contradicting the stability of M'. So $(1,1)$ edges can appear in ρ only when it goes from a hospital in a lower level to a resident in a higher level. So there can be at most $q-p$ many $(1,1)$ edges on ρ.

Induction step: Let the theorem hold for at most $i-1$ many $(-1,-1)$ edges. Let (h_k, r_{k+1}) be one such edge. Further, let $h_k \in H_a$ and $r_{k+1} \in R_b$. Consider the two subpaths $\rho_1 = \langle h_0, \ldots, r_k\rangle$ and $\rho_2 = \langle h_{k+1} \ldots, r_{t+1}\rangle$. As the number of $(-1,-1)$ edges in each of ρ_1 and ρ_2 is less than i, the induction hypothesis holds. Therefore, the number of $(1,1)$ edges in ρ_1 is at most $a-p$ plus the number of $(-1,-1)$ edges in ρ_1. Similarly, the number of $(1,1)$ edges in ρ_2 is $q-b$ plus the number of $(-1,-1)$ edges in ρ_2. The number of $(-1,-1)$ edges in ρ is one more than the total number of $(-1,-1)$ edges in ρ_1 and ρ_2. Hence the number of $(1,1)$ edges in ρ is at most the number of $(-1,-1)$ edges in ρ plus $a-p+q-b-1$. As there is an edge between h_k and r_{k+1}, $b \ge a-1$ by Lemma 6 part 2. Thus $a-p+q-b-1 \le q-p$, which completes the proof. ◀

Theorem 10 shows that M is a popular matching amongst all the feasible matchings in G.

▶ **Theorem 10.** *Let N be any feasible matching in G and **corr** be an arbitrary correspondence function from N to M.*

1. *If ρ is an alternating cycle in the decomposition of $M \oplus N$, then $\Delta(M \oplus \rho, M, \boldsymbol{corr})$[1]$\le 0$.*
2. *If ρ is an alternating path in the decomposition of $M \oplus N$ with exactly one end-point matched in M, then $\Delta(M \oplus \rho, M, \boldsymbol{corr}) \le 0$.*

[1] Note that when comparing $M \oplus \rho$ with M, we use the restriction of the correspondence function **corr** used to compare N with M. With the abuse of notation we refer to the restriction also as **corr**.

3. *If ρ is an alternating path in the decomposition of $M \oplus N$ with both the end-points matched in M then $\Delta(M \oplus \rho, M, \boldsymbol{corr}) \leq 0$.*

Proof. We prove the three cases below.

1. Let ρ be an alternating cycle in $M \oplus N$. Further, let $(r, h) \in M$. Consider $\rho' = \rho \setminus \{(r, h)\}$ which is an alternating path from h to r. The path ρ' starts and ends at the same level. Hence the number of $(1, 1)$ edges on ρ' is at most the number of $(-1, -1)$ edges on ρ'. The same holds for ρ.
2. Let ρ be an alternating path in $M \oplus N$ with exactly one end-point matched in M. Thus ρ has even length, and both its end-points are either hospitals or both are residents.

   Consider the first case. So let $\rho = \langle h_0, r_1, h_1, \ldots, r_t, h_t \rangle$ where $(r_i, h_i) \in M$ for all i. Thus $|M(h_0)| < |N(h_0)| \leq q^+(h_0)$, and hence h_0 is under-subscribed. Then by part 3 of Lemma 6 and feasibility of M, $h_0 \notin H_0$. By feasibility of N, $h_t \in H_0 \cup H_1$. As $(r_t, h_t) \in M$, $r \in R_0 \cup R_1$ by the definition of levels. Consider the subpath $\rho' = \rho \setminus \{(r_t, h_t)\}$ i.e. the path obtained by removing the edge (r_t, h_t) from ρ. Applying Theorem 9 to ρ' with $p \geq 1$ and $q = 0$ or $q = 1$, we get the number of $(1, 1)$ edges on ρ' to be at most the number of $(-1, -1)$ edges on ρ'.

   Consider the case when both the end-points of ρ are residents. Thus $\rho = \langle r_0, h_1, r_1, \ldots, h_t, r_t \rangle$ where $(h_i, r_i) \in M$ for all i. Again consider $\rho' = \rho \setminus \{(h_t, r_t)\}$. As r_0 is unmatched in M, $r_0 \in R_0$ by the definition of levels. Applying Theorem 9 to ρ' with $q = 0$, we get that the number of $(1, 1)$ edges on ρ' is at most the number of $(-1, -1)$ edges on ρ'.
3. Consider the case when both the end-points of the alternating path ρ are matched in M. Thus one end-point of ρ is a hospital whereas the other end-point is a resident. Let $\rho = \langle r_0, h_0, \ldots, r_t, h_t \rangle$ where $(r_i, h_i) \in M$ for all i. Hence $|M(h_t)| > |N(h_t)| \geq q^-(h_t)$ by feasibility of N. Therefore $h_t \in H_0 \cup H_1$ by part 6 of Lemma 6 which implies that $r_t \in R_0 \cup R_1$. Consider the subpath $\rho' = \rho \setminus \{(r_0, h_0), (r_t, h_t)\}$. Thus ρ' begins at h_0 and ends at r_t. Applying Theorem 9 with $q = 1$ and $0 \leq p \leq \ell$ gives that the number of $(1, 1)$ edges on ρ', and hence on ρ, is at most one more than the number of $(-1, -1)$ edges on ρ. These votes in favor of N are compensated by the end-points r_0 and h_t as r_0 is unmatched in N and $|M(h_t)| > |N(h_t)|$.

This completes the proof of the theorem. ◀

The following lemma proves that M is a maximum cardinality popular matching in G. The proof appears in the full-version [14].

▶ **Lemma 11.** *Let N be any feasible matching in G such that $|N| > |M|$. For any arbitrary correspondence function $\boldsymbol{corr}$ from N to M, $\Delta(N, M, \boldsymbol{corr}) < 0$.*

Size of G': Note that $|\mathcal{H}'| = \ell \cdot |\mathcal{H}|$. Furthermore, $|\mathcal{R}'| = |\mathcal{R}| + (\ell - 1) \cdot |\mathcal{H}|$. The second term in $|\mathcal{R}'|$ accounts for the number of dummy residents in G'. Finally, $|E'| \leq \ell \cdot |E| + 2(\ell - 1) \cdot |\mathcal{H}|$. The first term in $|E'|$ is because the preference list of every hospital in G appears ℓ times in G'. The second term is because every dummy resident in G' appears on the preference list of at most two hospitals in G'. Since, ℓ is upper bounded by $|\mathcal{R}|$, the size of our HR instance G' is $O(|\mathcal{R}| \cdot (|\mathcal{R}| + |\mathcal{H}| + |E|))$. This is the same as the running time of our algorithm to compute a maximum cardinality popular matching in G.

5 Popular matching amongst maximum cardinality feasible matchings

In this section our goal is to compute a maximum cardinality feasible matching that is popular amongst the set of maximum cardinality feasible matchings. Our algorithm is similar to the one described in Section 3. We give an overview of our algorithm and illustrate the execution of the same on the example instance in Figure 1.

As in Section 3, the algorithm here is a three phase algorithm. Phase-0 and Phase-2 are exactly as in Section 3. The modification is in Phase-1. In Section 3, we gave every hospital a second chance to propose if it was under-subscribed in the output of Phase-0. Here, we give $(|\mathcal{R}|-1)$ chances for every hospital to propose if it is under-subscribed at the end of Phase-0. To see this, recall the example instance in Figure 1. The output of hospital-proposing Gale-Shapley algorithm with all hospitals at *level*-0 is the matching $M_0 = \{(r_1, h_1^0), (r_2, h_2^0)\}$. When all the under-subscribed hospitals are allowed to propose at *level*-1, the matching obtained is $M_1 = \{(r_3, h_1^0), (r_2, h_2^0), (r_1, h_3^1)\}$. Note that h_4^1 and h_5^1 are still under-subscribed. The execution continues with under-subscribed hospitals being allowed to propose at level-2 and then at level-3 (see Figure 2 (d), Appendix A for a detailed proposal sequence). The matching obtained is $M_3 = \{(r_1, h_4^3), (r_2, h_3^2), (r_3, h_1^1), (r_4, h_2^0)\}$. Note that this is a maximum cardinality matching but it is infeasible as h_5 is deficient.

Subsequently, the only deficient hospital h_5 is allowed to propose at *level*-4. This results in h_5^4 applying to r_1, and thus M_3 gets changed to a feasible matching with maximum cardinality, which is $M_4 = \{(r_1, h_5^4), (r_2, h_3^2), (r_3, h_1^1), (r_4, h_2^0)\}$. A possible proposal sequence can be found in table in Figure 2 (b) followed by Figure 2 (d) in Appendix A.

To simulate this algorithm, we again reduce the HRLQ instance G to an HR instance G'' where every hospital has $\ell = |\mathcal{R}| + \sum_{h\in\mathcal{H}} q^-(h)$ many copies. In the interest of space, we give the details of the reduction and proof of correctness in the full-version [14]. A similar calculation for size of G'' as in Section 4 and the fact that $\ell \leq |\mathcal{R}|^2$ gives us the size of G'' as $O(|\mathcal{R}|^2 \cdot (|\mathcal{R}| + |\mathcal{H}| + |E|))$. This matches the running time of our algorithm in this section.

References

1 Péter Biró, Tamás Fleiner, Robert W. Irving, and David F. Manlove. The College Admissions Problem with Lower and Common Quotas. *Theoretical Computer Science*, 411(34-36):3136–3153, 2010.

2 Péter Biró, Robert W. Irving, and David Manlove. Popular Matchings in the Marriage and Roommates Problems. In *Proceedings of 7th International Conference on Algorithms anc Complexity*, pages 97–108, 2010.

3 Florian Brandl and Telikepalli Kavitha. Popular Matchings with Multiple Partners. *CoRR*, abs/1609.07531 (To appear in Proceedings of the 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science), 2016.

4 Ágnes Cseh and Telikepalli Kavitha. Popular Edges and Dominant Matchings. In *Proceedings of the 18th Conference on Integer Programming and Combinatorial Optimization*, pages 138–151, 2016.

5 David Gale and Llyod Shapley. College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69:9–14, 1962.

6 David Gale and Marilda Sotomayor. Some Remarks on the Stable Matching Problem. *Discrete Applied Mathematics*, 11(3):223–232, 1985.

7 Peter Gärdenfors. Match making: Assignments based on bilateral preferences. *Behavioral Science*, 20(3):166–173, 1975.

8 Koki Hamada, Kazuo Iwama, and Shuichi Miyazaki. The Hospitals/Residents Problem with Lower Quotas. *Algorithmica*, 74(1):440–465, 2016.

9 M. Hirakawa, Y. Yamauchi, S. Kijima, and M. Yamashita. On The Structure of Popular Matchings in The Stable Marriage Problem – Who Can Join a Popular Matching? In *Proceedings of the 3rd International Workshop on Matching Under Preferences*, 2015.

10 Chien-Chung Huang and Telikepalli Kavitha. Popular Matchings in the Stable Marriage Problem . *Information and Computation*, 222:180–194, 2013.

11 Telikepalli Kavitha. A Size-Popularity Tradeoff in the Stable Marriage Problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

12 Telikepalli Kavitha. Popular Half-Integral Matchings. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming*, pages 22:1–22:13, 2016.

13 Zoltán Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60(1):3–20, 2011. doi:10.1007/s00453-009-9371-7.

14 Meghana Nasre and Prajakta Nimbhorkar. Popular Matching with Lower Quotas. *CoRR*, abs/1704.07546, 2017. URL: http://arxiv.org/abs/1704.07546.

15 Meghana Nasre and Amit Rawat. Popularity in the Generalized Hospital Residents Setting. In *Proceedings of the 12th International Computer Science Symposium in Russia*, pages 245–259, 2017.

16 Alvin E. Roth. The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

17 Alvin E. Roth. On the Allocation of Residents to Rural Hospitals: A General Property of Two-Sided Matching Markets. *Econometrica*, 54(2):425–427, 1986.

18 Yu Yokoi. Envy-Free Matchings with Lower Quotas. *CoRR*, abs/1704.04888 (To appear in Proceedings of the 28th International Symposium on Algorithms and Computation), 2017. URL: http://arxiv.org/abs/1704.04888.

A Execution Sequence of Algorithm in Section 3

r_1 : h_1, h_3, h_4, h_5
r_2 : h_2, h_1, h_3
r_3 : h_2, h_1
r_4 : h_2

$(0,1)$ h_1 : r_1, r_2, r_3
$(0,1)$ h_2 : r_2, r_3, r_4
$(0,1)$ h_3 : r_1, r_2
$(0,1)$ h_4 : r_1
$(1,1)$ h_5 : r_1

(a)

Proposal	A/R	mMatching
$h_1^0 \to r_1$	✓	$\mathrm{m}\{(r_1, h_1^0)\}$
$h_2^0 \to r_2$	✓	$\mathrm{m}\{(r_1, h_1^0), (r_2, h_2^0)\}$
$h_3^0 \to r_1$	×	$\mathrm{m}\{(r_1, h_1^0), (r_2, h_2^0)\}$
$h_4^0 \to r_1$	×	$\mathrm{m}\{(r_1, h_1^0), (r_2, h_2^0)\}$
$h_5^0 \to r_1$	×	$\mathrm{m}\{(r_1, h_1^0), (r_2, h_2^0)\}$ $= M_0$
$h_3^1 \to r_1$	✓	$\mathrm{m}\{(r_1, h_3^1), (r_2, h_2^0)\}$
$h_4^1 \to r_1$	×	$\mathrm{m}\{(r_1, h_3^1), (r_2, h_2^0)\}$
$h_5^1 \to r_1$	×	$\mathrm{m}\{(r_1, h_3^1), (r_2, h_2^0)\}$
$h_1^0 \to r_2$	×	$\mathrm{m}\{(r_1, h_3^1), (r_2, h_2^0)\}$
$h_1^0 \to r_3$	✓	$\mathrm{m}\{(r_1, h_3^1), (r_2, h_2^0), (r_3, h_1^0)\}$ $= \mathbf{M_1}$

(b)

Proposal	A/R	mMatching
$h_5^2 \to r_1$	✓	$\mathrm{m}\{(r_1, h_5^2), (r_2, h_2^0), (r_3, h_1^0)\}$
$h_3^1 \to r_2$	✓	$\mathrm{m}\{(r_1, h_5^2), (r_2, h_3^1), (r_3, h_1^0)\}$
$h_2^0 \to r_3$	✓	$\mathrm{m}\{(r_1, h_5^2), (r_2, h_3^1), (r_3, h_2^0)\}$
$h_1^1 \to r_1$	×	$\mathrm{m}\{(r_1, h_5^2), (r_2, h_3^1), (r_3, h_2^0)\}$
$h_1^1 \to r_2$	✓	$\mathrm{m}\{(r_1, h_5^2), (r_2, h_1^1), (r_3, h_2^0)\}$ $= \mathbf{M_2}$

(c)

Proposal	A/R	mMatching
$h_4^2 \to r_1$	✓	$\mathrm{m}\{(r_1, h_4^2), (r_2, h_2^0), (r_3, h_1^0)\}$
$h_5^2 \to r_1$	×	$\mathrm{m}\{(r_1, h_4^2), (r_2, h_2^0), (r_3, h_1^0)\}$
$h_3^1 \to r_2$	✓	$\mathrm{m}\{(r_1, h_4^2), (r_2, h_3^1), (r_3, h_1^0)\}$
$h_2^0 \to r_3$	✓	$\mathrm{m}\{(r_1, h_4^2), (r_2, h_3^1), (r_3, h_2^0)\}$
$h_1^1 \to r_1$	×	$\mathrm{m}\{(r_1, h_4^2), (r_2, h_3^1), (r_3, h_2^0)\}$
$h_1^1 \to r_2$	✓	$\mathrm{m}\{(r_1, h_4^2), (r_2, h_1^1), (r_3, h_2^0)\}$
$h_3^2 \to r_1$	✓	$\mathrm{m}\{(r_1, h_3^2), (r_2, h_1^1), (r_3, h_2^0)\}$
$h_4^3 \to r_1$	✓	$\mathrm{m}\{(r_1, h_4^3), (r_2, h_1^1), (r_3, h_2^0)\}$
$h_3^2 \to r_2$	✓	$\mathrm{m}\{(r_1, h_4^3), (r_2, h_3^2), (r_3, h_2^0)\}$
$h_1^1 \to r_3$	✓	$\mathrm{m}\{(r_1, h_4^3), (r_2, h_3^2), (r_3, h_1^1)\}$
$h_2^0 \to r_4$	✓	$\mathrm{m}\{(r_1, h_4^3), (r_2, h_3^2), (r_3, h_1^1), (r_4, h_2^0)\}$ $= \mathbf{M_3}$
$h_5^3 \to r_1$	✓	$\mathrm{m}\{(r_1, h_5^3), (r_2, h_3^2), (r_3, h_1^1), (r_4, h_2^0)\}$ $= \mathbf{M_4}$

(d)

Figure 2 (a) Preference lists of residents and hospitals are recalled from Figure 1. Tables (b) (c) and (d) above show the proposal sequence for the instance in Figure 1. In each of the tables, the ✓ denotes that the proposal is accepted and the × denotes that the proposal is rejected. A possible proposal sequence for algorithm in Section 3 can be obtained by using table (b) followed by the sequence in table (c). A possible proposal sequence for algorithm in Section 5 can be obtained by using Table (b) followed by the sequence in Table (d).

The Complexity of the Diagonal Problem for Recursion Schemes*

Paweł Parys

University of Warsaw, Poland

Abstract

We consider nondeterministic higher-order recursion schemes as recognizers of languages of finite words or finite trees. We establish the complexity of the diagonal problem for schemes: given a set of letters A and a scheme $\mathcal{G}$, is it the case that for every number n the scheme accepts a word (a tree) in which every letter from A appears at least n times. We prove that this problem is $(m-1)$-EXPTIME-complete for word-recognizing schemes of order m, and m-EXPTIME-complete for tree-recognizing schemes of order m.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases diagonal problem, higher-order recursion schemes, intersection types, downward closure

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.45

1 Introduction

The *diagonal problem* in its original formulation over finite words asks, for a set of letters A and a language of words L, whether for every $n \in \mathbb{N}$ there is a word in L where every letter from A occurs at least n times. The same problem can be also considered for a language of finite trees. In this paper, we study the complexity of the diagonal problem for languages of finite words and finite trees recognized by nondeterministic higher-order recursion schemes.

Higher-order recursion schemes (*schemes* in short) are used to faithfully represent the control flow of programs in languages with higher-order functions. This formalism is equivalent via direct translations to simply-typed λY-calculus [22] and to higher-order OI grammars [8, 16]. Collapsible pushdown systems [9] and ordered tree-pushdown systems [5] are other equivalent formalisms. Schemes cover some other models such as indexed grammars [1] and ordered multi-pushdown automata [3].

The goal of this paper is to establish the complexity of the diagonal problem for higher-order schemes. By a recent result by Clemente et al. [6] we know that this problem is decidable. For schemes of order m their algorithm works in $f(m)$-fold exponential time for some quadratic function f (although the complexity of the algorithm is not mentioned explicitly in the paper, it can be easily estimated as being such). In the current work, we tighten the upper bound: we prove that the diagonal problem for word-recognizing (tree-recognizing) schemes of order m is $(m-1)$-EXPTIME-complete (m-EXPTIME-complete, respectively).

Let us recall from [6] that the decidability result for the diagonal problem entailed other decidability results for recursion schemes, concerning in particular computability of the downward closure of recognized languages [23], and the problem of separability by piecewise testable languages [7]. Although our complexity result for the diagonal problem does not

* Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

influence directly our knowledge on the complexity of the other problems (the aforementioned reductions preserve only decidability, but not complexity), it can be seen as the first step in establishing the complexity of the other problems as well.

Our solution is based on an appropriate system of intersection types. Intersection types were intensively used in the context of schemes, for several purposes like model-checking [11, 14, 4, 21], pumping [12], transformations of schemes [13, 6], etc. Among such type systems we have to distinguish those [18, 19], in which the (appropriately defined) size of a type derivation for a term approximates some quantity visible in the Böhm tree of that term. In particular, in our recent work [19] we have developed a type system that allows to solve the diagonal problem for a special case of a single-letter alphabet.

Here, we generalize the last type system mentioned above to multiple letters. In result, a type derivation in this system is labeled by flags of different kinds. The key property lies in some (quite rough) correspondence between words (trees) that can be generated from a term and type derivations for the term, where, for every letter a, the number of appearances of a in the generated word (tree) is approximated by the number of appearances of an appropriate flag in the type derivation. In effect, the diagonal problem reduces to checking whether there exist type derivations with arbitrarily many flags corresponding to every letter from the input set A.

Some further work was needed to carefully optimize the developed type system in order to obtain an algorithm achieving the optimal complexity.

Our paper is structured as follows. In Section 2 we introduce all necessary definitions. In Section 3 we introduce the type system, and we show how to use it for deciding the diagonal problem for word-recognizing schemes. Finally, Section 4 presents extensions of the algorithm, in particular to tree-recognizing schemes.

2 Preliminaries

Infinitary λ-calculus. The set of *sorts* (a/k/a simple types), constructed from a unique basic sort o using a binary operation $\to$, is defined as usual. We omit brackets on the right of an arrow, so e.g. $o\to(o\to o)$ is abbreviated to $o\to o\to o$. The order of a sort is defined by induction: $ord(o) = 0$, and $ord(\alpha_1\to\ldots\to\alpha_s\to o) = 1 + \max(ord(\alpha_1),\ldots,ord(\alpha_s))$ for $s \geq 1$.

A sort $\alpha_1\to\ldots\to\alpha_s\to o$ is *homogeneous* if $ord(\alpha_1) \geq \cdots \geq ord(\alpha_s)$ and all $\alpha_1,\ldots,\alpha_s$ are homogeneous. In the sequel we restrict ourselves to homogeneous sorts (even if not always this is written explicitly).

Let Σ be an infinite set of symbols (alphabet). To denote nondeterministic choices we use a symbol nd. Assuming that $\mathsf{nd} \notin \Sigma$, we denote $\Sigma^{\mathsf{nd}} = \Sigma \cup \{\mathsf{nd}\}$. Let also $\mathcal{V} = \{x^\alpha, y^\beta, z^\gamma, \ldots\}$ be a set of variables, containing infinitely many variables of every homogeneous sort (sort of a variable is written in superscript).

We consider infinitary, sorted λ-calculus. *Infinitary λ-terms* (or just *λ-terms*) are defined by coinduction, according to the following rules:

- node constructor – if $a \in \Sigma^{\mathsf{nd}}$, and $P_1^o,\ldots,P_r^o$ are λ-terms, then $(a\langle P_1^o,\ldots,P_r^o\rangle)^o$ is a λ-term,[1]

[1] Our node constructor differs from the standard definition in two aspects. First, one usually assumes that symbols are ranked, i.e., that the number r is determined by the choice of a. Second, typically a symbol a is considered itself as a λ-term of sort $\underbrace{o\to\ldots\to o}_{r}\to o$, which after applying $P_1^o,\ldots,P_r^o$ as arguments is equivalent to our $(a\langle P_1^o,\ldots,P_r^o\rangle)^o$. These are, though, only superficial differences.

- variable – every variable $x^\alpha \in \mathcal{V}$ is a λ-term,
- application – if $P^{\alpha\to\beta}$ and Q^α are λ-terms, then $(P^{\alpha\to\beta}\, Q^\alpha)^\beta$ is a λ-term, and
- λ-binder – if P^β is a λ-term and x^α is a variable, then $(\lambda x^\alpha.P^\beta)^{\alpha\to\beta}$ is a λ-term;

in the above, α, β, and $\alpha\to\beta$ are homogeneous sorts. We naturally identify λ-terms differing only in names of bound variables. We often omit the sort annotations of λ-terms, but we keep in mind that every λ-term (and every variable) has a particular sort. *Free variables* of a λ-term are defined as usual. A λ-term P is *closed* if it has no free variables.

For a λ-term P, the *order* of P is just the order of its sort, while the *complexity* of P is the smallest number m such that the order of all subterms of P is at most m. We restrict ourselves to λ-terms that have finite complexity. We also define the order of a β-reduction as the order of the involved variable. More precisely, for a number $k \in \mathbb{N}$, we say that there is a β-reduction of order k from a λ-term P to a λ-term Q, written $P \to_{\beta(k)} Q$, if Q is obtainable from P by replacing a redex $(\lambda x.R)\, S$ where $ord(x) = k$ with $R[S/x]$.

Trees. A *tree* is defined as a λ-term that is built using only node constructors, i.e., not using variables, applications, nor λ-binders. A tree is Γ-*labeled* if only symbols from Γ appear in it.

Let us now define how we resolve nondeterministic choices. Although this is mainly used for trees, we define it for arbitrary λ-terms. We write $P \to_{\mathsf{nd}} Q$ if Q is obtained from P by choosing some appearance of the nd symbol surrounded only by symbols from Σ, and removing this nd symbol together with all but one of its arguments. Formally, we let $\to_{\mathsf{nd}}$ to be the smallest relation such that $\mathsf{nd}\langle P_1, \dots, P_r\rangle \to_{\mathsf{nd}} P_i$ for $i \in \{1, \dots, r\}$, and if $a \in \Sigma$, and $P_i \to_{\mathsf{nd}} P_i'$ for some $i \in \{1, \dots, r\}$, and $P_j = P_j'$ for all $j \in \{1, \dots, r\} \setminus \{i\}$, then $a\langle P_1, \dots, P_r\rangle \to_{\mathsf{nd}} a\langle P_1', \dots, P_r'\rangle$. For a relation r, by r^* we denote the reflexive transitive closure of r. For a λ-term P (which is usually a Σ^{nd}-labeled, potentially infinite tree), by $\mathcal{L}(P)$ we denote the set of all finite, Σ-labeled trees T such that $P \to_{\mathsf{nd}}^* T$.

Böhm Trees. We consider Böhm trees only for closed λ-terms of sort o. For such a term P, its *Böhm tree* $BT(P)$ is constructed by coinduction, as follows: if there is a sequence of β-reductions from P to a λ-term of the form $a\langle P_1, \dots, P_r\rangle$ (where $a \in \Sigma^{\mathsf{nd}}$), then $BT(P) = a\langle BT(P_1), \dots, BT(P_r)\rangle$; otherwise $BT(P) = \mathsf{nd}\langle\rangle$.

Higher-Order Recursion Schemes. We use a very loose definition of schemes. A *higher-order recursion scheme* (or just a *scheme*) is a triple $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0^o)$, where $\mathcal{N} \subseteq \mathcal{V}$ is a finite set of nonterminals, $\mathcal{R}$ is a function that maps every nonterminal $N \in \mathcal{N}$ to a finite λ-term whose free variables are contained in $\mathcal{N}$ and whose sort equals the sort of N, and $N_0^o \in \mathcal{N}$ is a starting nonterminal, being of sort o. We assume that elements of $\mathcal{N}$ are not used as bound variables, and that $\mathcal{R}(N)$ is not a nonterminal. The order of the scheme is defined as the maximum of complexities of $\mathcal{R}(N)$ over all its nonterminals N.

The infinitary λ-term *generated by* a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0^o)$, denoted $\Lambda(\mathcal{G})$, is defined as the limit of the following process starting from N_0^o: take any nonterminal N appearing in the current term, and replace it by $\mathcal{R}(N)$ (so that every nonterminal is eventually replaced). Observe that in the limit we obtain a closed λ-term of sort o and of complexity not greater than the order of the scheme. The *language of* $\mathcal{G}$ is defined as $\mathcal{L}(\mathcal{G}) = \mathcal{L}(BT(\Lambda(\mathcal{G})))$.

We remark that according to our definition all subterms of all λ-terms (and all nonterminals as well) have homogeneous sorts; usually it is not assumed that sorts used in a scheme are homogeneous. It is, however, the case that any scheme using also non-homogeneous sorts

can be converted into one in which all sorts are homogeneous, and that this can be done in logarithmic space [20]. We make the homogeneity assumption for technical convenience.

A *word* is defined as a tree in which every node has at most one child (such a tree can be identified with a word understood in the classic sense). We say that a λ-term P is *word-recognizing* if for every its subterm of the form $a\langle P_1, \ldots, P_r\rangle$ with $a \in \Sigma$ it holds $r \leq 1$; a scheme $\mathcal{G}$ is *word-recognizing* if $\Lambda(\mathcal{G})$ is word-recognizing. In this case, all elements of $\mathcal{L}(BT(P))$ or $\mathcal{L}(\mathcal{G})$, respectively, are words.

▶ **Example 1.** Consider the higher-order recursion scheme $\mathcal{G}_1$ with two nonterminals, M^o (taken as the starting nonterminal) and $\mathsf{N}^{(o\to o)\to o}$, and with rules

$$\mathcal{R}(\mathsf{M}) = \mathsf{N}\,(\lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle)\,, \qquad \mathcal{R}(\mathsf{N}) = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), \mathsf{N}\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle\,.$$

We obtain $\Lambda(\mathcal{G}_1) = R_1\,(\lambda \mathsf{x}.\mathsf{nd}\langle \mathsf{a}\langle \mathsf{x}\rangle, \mathsf{b}\langle \mathsf{x}\rangle\rangle)$, where R_1 is the unique λ-term such that $R_1 = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle$. We have $BT(\Lambda(\mathcal{G}_1)) = \mathsf{nd}\langle T_{2^0}, \mathsf{nd}\langle T_{2^1}, \mathsf{nd}\langle T_{2^2}, \ldots\rangle\rangle\rangle$, where $T_0 = \mathsf{c}\langle\rangle$ and $T_{i+1} = \mathsf{nd}\langle \mathsf{a}\langle T_i\rangle, \mathsf{b}\langle T_i\rangle\rangle$. In $\mathcal{L}(\mathcal{G}_1)$ we have words of length $2^i + 1$ for all $i \in \mathbb{N}$, where the first 2^i letters are chosen from $\{\mathsf{a}, \mathsf{b}\}$ arbitrarily, and the last letter is c. In the following examples we will continue to consider this scheme, together with the set $A = \{\mathsf{a}, \mathsf{b}\}$.

3 Type System for the Diagonal Problem

In this section we introduce a type system that allows to solve the diagonal problem for schemes.

▶ **Definition 2.** For a set of trees L and a set of symbols A, the predicate $Diag_A(L)$ holds if for every $n \in \mathbb{N}$ there is some $T \in L$ with at least n occurrences of every symbol from A. The *diagonal problem* for tree-recognizing order-m schemes is to decide whether $Diag_A(\mathcal{L}(\mathcal{G}))$ holds, given a scheme $\mathcal{G}$ of order at most m and a set A. The diagonal problem for *word-recognizing* order-m schemes is as the above, but with the restriction that $\mathcal{G}$ is word-recognizing.

▶ **Theorem 3.** *For $m \geq 1$, the diagonal problem for word-recognizing order-$(m+1)$ schemes is m-EXPTIME-complete. For $m \in \{-1, 0\}$ it is NP-complete.*

Throughout the rest of this section we solve the diagonal problem for word-recognizing schemes, and thus all schemes considered here are assumed to be word-recognizing. Moreover, we fix a set of symbols A, for which we want to solve the diagonal problem.

Intuitions. The main novelty of our type system lies in labeling nodes of type derivations by two kinds of labels called flags and markers. To each marker we assign a number, called an order. Flags, beside of their order, are also identified by a symbol from A; thus we have (k, a)-flags for $k \in \mathbb{N}$ and $a \in A$. While deriving a type for a λ-term of complexity at most $m + 1$, we use markers of order from the range $0, \ldots, m$, and flags of order from the range $1, \ldots, m + 1$.

Let P_{m+1} be a λ-term of complexity at most $m + 1$. Recall that our goal is to describe a word $T \in \mathcal{L}(BT(P_{m+1}))$ using a type derivation for P_{m+1} itself. While doing that, we want to preserve the information that T has many appearances of every symbol from A.

Since T can be found in some finite prefix of $BT(P_{m+1})$, in order to find T it is enough to perform finitely many β-reductions from P_{m+1}. Moreover, thanks to the fact that all sorts are homogeneous, the β-reductions can be rearranged so that those of higher order are performed first. Namely, we can find λ-terms $P_0, \ldots, P_m$ such that

$$P_{m+1} \to^*_{\beta(m)} P_m \to^*_{\beta(m-1)} \ldots \to^*_{\beta(0)} P_0 \qquad \text{and} \qquad P_0 \to^*_{\mathsf{nd}} T\,.$$

Some prefix of P_0 can be seen as a tree, in which we can find a path on which there are all symbols of T, and some additional nd symbols. Let us place an order-0 marker in the leaf ending this path. Additionally, for every symbol $a \in A$, we place $(1,a)$-flags in all a-labeled nodes of the considered path.

Next, we proceed back to P_1. The leaf constructor of P_0 containing our order-0 marker was created out of some particular appearance of such constructor in P_1; let us put there as well the order-0 marker. Similarly, we find node constructors in P_1 out of which in P_0 we obtain node constructors with flags, and we transfer the flags back to P_1. The crucial observation is that no two flagged node constructors of P_0 could come out of a single node constructor of P_1. Indeed, recall that all the β-reductions between P_1 and P_0 are of order 0. This means that in every such β-reduction we take a whole subtree (i.e., a λ-term of sort o) of P_1, and we replace it somewhere, possibly replicating it. But since all flags lie in P_0 on a single path, they may lie only in at most one copy of the replicated subtree. In effect, the number of appearances of order-1 flags of every kind is the same in P_1 as in P_0.

We cannot directly repeat the same reasoning to move flags from P_1 back to P_2, since now there is a problem: a single node constructor in P_2 may result in multiple (uncontrollably many) node constructors with a flag in P_1. We rescue ourselves by considering only $|A|$ paths in P_1, one for each symbol in A. Namely, for every symbol $a \in A$ we place in P_1 a marker of order 1, choosing in this way the path from the root to the position of this marker. Then, for every node labeled by a $(1,a)$-flag we place a $(2,a)$-flag in the closest ancestor that lies on the chosen path. Although the number of $(2,a)$-flags may be smaller than the number of $(1,a)$-flags (the closest ancestor on the path may be the same for multiple $(1,a)$-flags), we can ensure that it is smaller only logarithmically; to do so, we choose the marked node in a clever way: staring from the root, we always proceed to this subtree in which the number of $(1,a)$-flags is the largest. In effect, if the number of $(1,a)$-flags was "very large", then also the number of $(2,a)$-flags will be "very large".

Once for every $a \in A$ all $(2,a)$-flags lie on a single path of P_1, we can transfer them back to P_2 without changing their number. Then in P_2 we again reduce to $|A|$ paths by introducing markers of order 2, and so on. At the end we obtain some labeling of P_{m+1} by several kinds of flags and markers. The goal of the type system we develop is, roughly speaking, to ensure that a labeling of P_{m+1} actually is obtainable in the process as above (in fact, we will not be labeling nodes of P_{m+1} itself, but rather nodes of a type derivation for P_{m+1}).

Type Judgments. Recall that A is the set of symbols for which we want to solve the diagonal problem. For storing the information about flags and markers used in a derivation of a type we use flag sets and marker multisets. Recall that a flag is parameterized by a pair (k,a), where $k \in \mathbb{N}$ is called an order, and $a \in A$ is called a symbol. For flags it is enough to remember for every order whether at least one flag of this order was used, and if so, then also a symbol of this flag (if flags with multiple symbols were used, it is enough for us to remember just one of these symbols). Thus for $m \in \mathbb{N}$ we define $\mathcal{F}_m$ to contain sets $F \subseteq \{1,\dots,m\} \times A$ such that $(k,a),(k,b) \in F$ implies $a = b$. Such sets F are called *m-bounded flag sets*. For markers the situation is slightly different, as we want to remember precisely how many markers were used. Moreover, markers do not have a symbol, only an order. We thus define $\mathcal{M}_m$ to contain functions $M\colon \mathbb{N} \to \{0,\dots,|A|\}$ such that $M(0) \leq 1$ and $M(k) = 0$ for all $k > m$. Such functions M are called *m-bounded marker multisets*.

By $M + M'$ and $M - M'$ we mean the coordinatewise sum or difference, respectively. We use $\mathbf{0}$ to denote a function that maps every element of its domain to 0 (where the domain

should be always clear from the context). By $\{\!\{k_1, \ldots, k_n\}\!\}$ we mean the multiset M such that $M(k) = |\{i \in \{1, \ldots, n\} \mid k_i = k\}|$ for all $k \in \mathbb{N}$. When $F \in \mathcal{F}_m$, $M \in \mathcal{M}_m$, $n \in \mathbb{N}$, and $\Box$ is one of $\leq$, $>$, we write $F\restriction_{\Box n}$ for $\{(k, a) \in F \mid k \Box n\}$, and $M\restriction_{\Box n}$ for the function that maps every k to $M(k)$ if $k \Box n$, and to 0 if $\neg(k \Box n)$.

Next, for every sort α and for $m \in \mathbb{N}$ we define three sets: the set $\mathcal{T}^\alpha$ of *types* of sort α, the set $\mathcal{TT}^\alpha_m$ of *m-bounded type triples* of sort α, and the set $\mathcal{TC}^\alpha$ of *triple containers* of sort α. They are defined by mutual induction on the structure of α.

If $\alpha = \alpha_1 \to \ldots \to \alpha_s \to o$, the set $\mathcal{T}^\alpha$ contains types that are of the form $C_1 \to \ldots \to C_s \to o$, where $C_i \in \mathcal{TC}^{\alpha_i}$ for $i \in \{1, \ldots, s\}$.

Type triples in $\mathcal{TT}^\alpha_m$ are just triples $(F, M, C_1 \to \ldots \to C_s \to o) \in \mathcal{F}_m \times \mathcal{M}_m \times \mathcal{T}^\alpha$, where $M(k) = 0$ for all $(k, a) \in F$, and where $M(0) + \sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) = 1$ (we will define $\mathsf{Mk}(C_i)$ soon). These triples store a type, together with the information about flags and markers used while deriving this type. In order to distinguish type triples from types, the former are denoted by letters with a hat, like $\hat{\tau}$. We also define a function Mk that extracts the marker multiset out of a type triple: $\mathsf{Mk}(\hat{\tau}) = M$ for $\hat{\tau} = (F, M, \tau)$. A type triple is *balanced* if $\mathsf{Mk}(\hat{\tau}) = \mathbf{0}$; otherwise it is *unbalanced*.

Triple containers are used to store (multi)sets of type triples that have to be derived for an argument of a λ-term, or for a λ-term substituted for a free variable. For balanced type triples, triple containers behave like sets, that is, they remember only whether every balanced type triple is required or not. Conversely, for unbalanced type triples, triple containers behave like multisets, that is, they remember precisely how many times every unbalanced type triple is required. Thus, formally, in $\mathcal{TC}^\alpha$ we have functions $C \colon \mathcal{TT}^\alpha_{ord(\alpha)} \to \{0, \ldots, |A|\}$ such that $C(\hat{\tau}) \leq 1$ if $\mathsf{Mk}(\hat{\tau}) = \mathbf{0}$. For $C \in \mathcal{TC}^\alpha$ we define $\mathsf{Mk}(C) = \sum_{\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}} \sum_{i=1}^{C(\hat{\tau})} \mathsf{Mk}(\hat{\tau})$. For two triple containers $C, D \in \mathcal{TC}^\alpha$ we define their sum $C \sqcup D \colon \mathcal{TT}^\alpha_{ord(\alpha)} \to \mathbb{N}$ so that for every $\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}$,

$$(C \sqcup D)(\hat{\tau}) = \begin{cases} C(\hat{\tau}) + D(\hat{\tau}) & \text{if } \mathsf{Mk}(\hat{\tau}) \neq \mathbf{0}\,, \\ \max(C(\hat{\tau}), D(\hat{\tau})) & \text{if } \mathsf{Mk}(\hat{\tau}) = \mathbf{0}\,. \end{cases}$$

We also say that $C \sqsubseteq D$ if $C(\hat{\tau}) = D(\hat{\tau})$ for every unbalanced $\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}$, and $C(\hat{\tau}) \leq D(\hat{\tau})$ for every balanced $\hat{\tau} \in \mathcal{TT}^\alpha_{ord(\alpha)}$. We sometimes write $\{\!\{\hat{\tau}_1, \ldots, \hat{\tau}_n\}\!\}$ or $\{\!\{\hat{\tau}_i \mid i \in \{1, \ldots, n\}\}\!\}$ to denote the triple container C such that $C(\hat{\sigma}) = |\{i \in \{1, \ldots, n\} \mid \hat{\tau}_i = \hat{\sigma}\}|$ for every unbalanced type triple $\hat{\sigma}$, and $C(\hat{\sigma}) = 1 \Leftrightarrow \exists i \in \{1, \ldots, n\}\,.\, \hat{\tau}_i = \hat{\sigma}$ for every balanced type triple $\hat{\sigma}$.

A *type environment* is a function Γ that maps every variable x^α to a triple container from $\mathcal{TC}^\alpha_{ord(\alpha)}$. We use ε to denote the type environment mapping every variable to $\mathbf{0}$. When $\Gamma(x) = \mathbf{0}$, by $\Gamma[x \mapsto C]$ we denote the type environment that maps x to C, and every other variable y to $\Gamma(y)$ (whenever we write $\Gamma[x \mapsto C]$, we implicitly require that $\Gamma(x) = \mathbf{0}$). For two type environments Γ, Γ' we define their sum $\Gamma \sqcup \Gamma'$ so that $(\Gamma \sqcup \Gamma')(x) = \Gamma(x) \sqcup \Gamma'(x)$ for every variable x.

A *type judgment* is of the form $\Gamma \vdash_m P : \hat{\tau} \triangleright c$, where Γ is a type environment, $m \in \mathbb{N}$ is called the *order* of the type judgment, P is a λ-term, $\hat{\tau}$ is an m-bounded type triple of the same sort as P (i.e. $\hat{\tau} \in \mathcal{TT}^\alpha_m$ when P is of sort α), and c is a function $A \to \mathbb{N}$ called a *flag counter*.

As usually for intersection types, the intuitive meaning of a type $C \to \tau$ is that a λ-term having this type can return a λ-term having type τ, while taking an argument for which we can derive all type triples from C. Let us now explain the meaning of a type judgment $\Gamma \vdash_m P : (F, M, \tau) \triangleright c$. Obviously τ is the type derived for P, and Γ contains type triples that could be used for free variables of P in the derivation. As explained above for triple

containers, balanced and unbalanced type triples behave differently: all unbalanced type triples assigned to variables by Γ have to be used exactly once in the derivation; conversely, balanced type triples may be used any number of times. Going further, the order m of the type judgment bounds the order of flags and markers that can be used in the derivation: flags can be of order at most $m+1$, and markers of order at most m. The multiset M counts markers used in the derivation, together with those provided by free variables (i.e., we imagine that some derivations, specified by the type environment, are already substituted in our derivation for free variables); we, however, do not include markers provided by arguments of the λ-term (i.e. coming from the triple containers C_i when $\tau = C_1 \to \ldots \to C_s \to o$). The set F contains an information about flags of order at most m used in the derivation. A pair (k,a) can be contained in F if a (k,a)-flag is placed in the derivation itself, or provided by a free variable, or provided by an argument. We do not have to keep in F all such pairs, i.e., if we can derive a type triple with some flag set F, then we can derive it also with every subset of F as the flag set. In fact, we cannot keep in F all such pairs due to two restrictions. First, the definition of a flag set allows to have in F at most one pair (k,a) for every order k. Second, we intentionally remove from F all pairs (k,a) for which $M(k) > 0$. Finally, in a type judgment we have a function c, called a *flag counter*, that for each symbol a counts the number of $(m+1,a)$-flags present in the derivation.

Type System. Before giving rules of the type system, let us state two general facts. First, all type derivations are assumed to be finite – although we derive types mostly for infinite λ-terms, each type derivation analyzes only a finite part of a term. Second, we require that premisses and conclusions of all rules are valid type judgments. For example, when the type environment appearing in the conclusion of a rule is $\Gamma \sqcup \Gamma'$, this implies that for all x and all unbalanced type triples $\hat{\tau}$ it holds $\Gamma(x)(\hat{\tau}) + \Gamma'(x)(\hat{\tau}) \leq |A|$ (so that $(\Gamma \sqcup \Gamma')(x)$ is indeed a valid triple container). Let us also remark that rules of the type system will guarantee that the order m of all type judgments used in a derivation will be the same.

Rules of the type system correspond to particular constructs of λ-calculus. We start by giving the first three rules:

$$\frac{M\restriction_{\leq ord(x)} = M'}{\varepsilon[x \mapsto \{\!\{(F,M',\tau)\}\!\}] \vdash_m x : (F,M,\tau) \triangleright \mathbf{0}}\ (\textsc{Var}) \qquad \frac{\Gamma \vdash_m P_i : \hat{\tau} \triangleright c \qquad i \in \{1,\ldots,r\}}{\Gamma \vdash_m \mathsf{nd}\langle P_1,\ldots,P_r\rangle : \hat{\tau} \triangleright c}\ (\textsc{Nd})$$

$$\frac{\Gamma[x \mapsto C'] \vdash_m P : (F,M,\tau) \triangleright c \qquad C' \sqsubseteq C}{\Gamma \vdash_m \lambda x.P : (F, M - \mathsf{Mk}(C), C \to \tau) \triangleright c}\ (\lambda)$$

The (Var) rule allows to have in the resulting marker multiset M some numbers that do not come from the multiset assigned to x by the type environment; these are the orders of markers placed in the leaf using this rule. Notice, however, that we allow here only orders greater than $ord(x)$. This is consistent with the intuitive description of the type system (page 4), which says that a marker of order k can be put in a place that will be a leaf after performing all β-reductions of order at least k. Indeed, the variable x remains a leaf after performing β-reductions of orders greater than $ord(x)$, but while performing β-reductions of order $ord(x)$ this leaf will be replaced by a subterm substituted for x. Recall also that, by definition of a type judgment, we require that $(F,M',\tau) \in \mathcal{TT}^{\alpha}_{ord(x)}$ and $(F,M,\tau) \in \mathcal{TT}^{\alpha}_{m}$, for appropriate sort α; this introduces a bound on maximal numbers that may appear in F and M.

▶ **Example 4.** Denoting $\hat{\rho}_0 = (\emptyset, \{\!\{0\}\!\}, o)$ we can derive:

$$\frac{}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!\{0\}\!\}, o) \triangleright \mathbf{0}}\ (\textsc{Var}) \qquad \frac{}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \mathbf{0}}\ (\textsc{Var})$$

In the derivation on the right, two markers of order 1 are placed in the conclusion of the rule.

We see that to derive a type for the nondeterministic choice $\mathsf{nd}\langle P_1, \dots, P_r\rangle$, we need to derive it for one of the subterms $P_1, \dots, P_r$.

For the (λ) rule, recall that $C' \sqsubseteq C$ means that in C' we have all unbalanced type triples from C, and some subset of balanced type triples from C. Thus in a subderivation concerning the λ-term P, we need to use all unbalanced type triples provided by an argument of $\lambda x.P$, while balanced type triples may be used or not. Recall also that we intend to store in the marker multiset the markers contained in the derivation itself and those provided by free variables, but not those provided by arguments. Because of this, in the conclusion of the rule we remove from M the markers provided by x. It is required, implicitly, that the result remains nonnegative. The set F, unlike M, stores also flags provided by arguments, so we do not need to remove anything from F.

▶ **Example 5.** In this example we show how the (ND) and (λ) rules can be used. Notice that in the conclusion of the (λ) rule, in both derivations, we remove 0 from the marker multiset, because an order-0 marker is provided by x.

$$\dfrac{\dfrac{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\{(1,\mathsf{a})\}, \{\!\{0\}\!\}, o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\{(1,\mathsf{a})\}, \{\!\{0\}\!\}, o) \triangleright \mathbf{0}}\,(\text{ND})}{\varepsilon \vdash_1 \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\{(1,\mathsf{a})\}, \mathbf{0}, \{\!\{\hat{\rho}_0\}\!\}{\to}o) \triangleright \mathbf{0}}\,(\lambda)$$

$$\dfrac{\dfrac{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}}\,(\text{ND})}{\varepsilon \vdash_1 \lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle : (\emptyset, \{\!\{1,1\}\!\}, \{\!\{\hat{\rho}_0\}\!\}{\to}o) \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}}\,(\lambda)$$

The next three rules use a predicate Comp_m, saying how flags and markers from premisses contribute to the conclusion. It takes "as input" pairs (F_i, c_i) for $i \in I$, consisting of a flag set F_i and a flag counter c_i from some premiss. Moreover, the predicate takes a marker multiset M that will appear in the conclusion of the rule. The goal is to compute a flag set F and a flag counter c that should be placed in the conclusion. First, for each $k \in \{1, \dots, m+1\}$ consecutively, we decide which flags of order k should be placed in the considered node of a type derivation. We follow here the rules mentioned in the intuitive description. Namely, we place a (k,a)-flag if we are on the path leading to a marker of order $k-1$ (i.e., if $M(k-1) > 0$), and simultaneously we receive an information about a $(k-1,a)$-flag. By receiving this information we mean that either a $(k-1,a)$-flag was placed in the current node, or $(k-1,a)$ belongs to some set F_i. Actually, we place multiple (k,a)-flags: one per each $(k-1,a)$-flag placed in the current node, and one per each set F_i containing $(k-1,a)$. Then, we compute F and c. In $c(a)$, for every $a \in A$, we store the number of $(m+1,a)$-flags: we sum all the flag counters c_i, and we add the number of $(m+1,a)$-flags placed in the current node. In F, we allow to keep elements of all F_i, and we allow to add pairs (k,a) for flags that were placed in the current node, but it can be chosen "nondeterministically" which of them are actually taken to F, and which are dropped. It is often necessary to drop some elements, since when the set F is used in a type triple, the definitions of a flag set and of a type triple put additional requirements on this set.

Below we give a formal definition, in which $f'_{k,a}$ contains the number of (k,a)-flags placed in the current node, while $f_{k,a}$ additionally counts the number of premisses for which $(k,a) \in F_i$. We say that $(F,c) \in \mathit{Comp}_m(M; ((F_i,c_i))_{i\in I})$ when

$$F \subseteq \{(k,a) \mid f_{k,a} > 0\}, \qquad \text{and} \qquad c(a) = f_{m+1,a} + \sum_{i\in I} c_i(a) \qquad \text{for all } a \in A,$$

where, for $k \in \{0, \dots, m+1\}$ and $a \in A$,

$$f_{k,a} = f'_{k,a} + \sum_{i \in I} |F_i \cap \{(k,a)\}|, \qquad f'_{k,a} = \begin{cases} 0 & \text{if } k = 0 \text{ or } M(k-1) = 0, \\ f_{k-1,a} & \text{otherwise.} \end{cases}$$

We now present rules for node constructors using symbols other than nd:

$$\frac{(F,c) \in \mathit{Comp}_m(M; (\{(0,a)\}, \mathbf{0})) \qquad a \neq \mathsf{nd}}{\varepsilon \vdash_m a\langle\rangle : (F,M,o) \triangleright c} \text{ (Con0)}$$

$$\frac{\Gamma \vdash_m P : (F',M,o) \triangleright c' \qquad (F,c) \in \mathit{Comp}_m(M; (\{(0,a)\}, \mathbf{0}), (F',c')) \qquad a \neq \mathsf{nd}}{\Gamma \vdash_m a\langle P\rangle : (F,M,o) \triangleright c} \text{ (Con1)}$$

In these rules we do not claim that the set $\{(0,a)\}$ passed to Comp_m is an element of $\mathcal{F}_m$ (and in fact it is not, because the order is 0, which is forbidden for flags; we also do not necessarily have that $a \in A$). The effect of passing this set is that if $M(0) > 0$ (i.e., we are on the path to the order-0 marker) and $a \in A$, then Comp_m places a $(1,a)$-flag in the current node, and maybe also some (k,a)-flags for higher k. In the (Con0) rule, i.e., if we are in a leaf, we are allowed to place markers of arbitrary order: the marker multiset M may be arbitrary.

▶ **Example 6.** The (Con1) rule may be instantiated in the following ways:

$$\frac{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!\{0\}\!\}, o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\{(1,\mathsf{a})\}, \{\!\{0\}\!\}, o) \triangleright \mathbf{0}} \text{ (Con1)}$$

$$\frac{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{x} : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \mathbf{0}}{\varepsilon[\mathsf{x} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{a}\langle\mathsf{x}\rangle : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}} \text{ (Con1)}$$

In the first example, a $(1,\mathsf{a})$-flag is placed in the conclusion of the rule (because the marker multiset contains 0, the pair $(0,\mathsf{a})$ passed to the Comp_1 predicate results in the $(1,\mathsf{a})$ pair in the flag set). In the second example, $(1,\mathsf{a})$- and $(2,\mathsf{a})$-flags are placed in the conclusion of the (Con1) rule: since order-1 markers are visible, we do not put $(1,\mathsf{a})$ to the flag set, but instead we create a $(2,\mathsf{a})$-flag, which results in increasing the flag counter.

The last rule describes application:

$$\frac{\begin{array}{c}\Gamma' \vdash_m P : (F', M', \{\!\{(F_i{\restriction}_{\leq ord(Q)}, M_i{\restriction}_{\leq ord(Q)}, \tau_i) \mid i \in I\}\!\} \to \tau) \triangleright c' \\ \Gamma_i \vdash_m Q : (F_i, M_i, \tau_i) \triangleright c_i \text{ for each } i \in I \qquad M = M' + \sum_{i \in I} M_i \qquad ord(Q) \leq m \\ (F,c) \in \mathit{Comp}_m(M; (F',c'), ((F_i{\restriction}_{>ord(Q)}, c_i))_{i \in I}) \qquad \{(k,a) \in F' \mid M(k) = 0\} \subseteq F\end{array}}{\Gamma' \sqcup \bigsqcup_{i \in I} \Gamma_i \vdash_m P\,Q : (F,M,\tau) \triangleright c} \text{ (@)}$$

In this rule, it is allowed (and potentially useful) that for two different $i \in I$ the type triples (F_i, M_i, τ_i) are equal. It is also allowed that $I = \emptyset$, in which case no type needs to be derived for Q. Observe how flags and markers coming from premisses concerning Q are propagated: only flags and markers of order $k \leq ord(Q)$ are visible to P, while only flags of order $k > ord(Q)$ are passed to the Comp_m predicate. This can be justified if we recall the intuitions staying behind the type system (see page 4). Indeed, while considering flags and markers of order k, we should imagine the λ-term obtained from the current λ-term by performing all β-reductions of order at least k; the distribution of flags and markers of order

k in the current λ-term actually simulates their distribution in this imaginary λ-term. Thus, if $k \leq ord(Q)$, then our application will disappear in this imaginary λ-term (thanks to the homogenity assumption), and Q will be already substituted somewhere in P; for this reason we need to pass the information about flags and markers of order k from Q to P. Conversely, if $k > ord(Q)$, then in the imaginary λ-term the considered application will be still present, and in consequence the subterm corresponding to P will not see flags and markers of order k placed in the subterm corresponding to Q. The condition $\{(k,a) \in F' \mid M(k) = 0\} \subseteq F$ (saying that some flags from F' cannot disappear) is useful for proofs.

▶ **Example 7.** Recalling that $\hat{\rho}_0 = (\emptyset, \{\!\{0\}\!\}, o)$, denote by $\hat{\tau}_{\mathsf{a}}$ and $\hat{\tau}_{\mathsf{m}}$ the type triples derived in Example 5: $\hat{\tau}_{\mathsf{a}} = (\{(1,\mathsf{a})\}, \mathbf{0}, \{\!\{\hat{\rho}_0\}\!\}{\to}o)$ and $\hat{\tau}_{\mathsf{m}} = (\emptyset, \{\!\{1,1\}\!\}, \{\!\{\hat{\rho}_0\}\!\}{\to}o)$. We can derive:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}\}\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_{\mathsf{a}} \triangleright \mathbf{0}} \qquad \dfrac{\dfrac{}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{m}}\}\!\}] \vdash_1 \mathsf{f} : \hat{\tau}_{\mathsf{m}} \triangleright \mathbf{0}} \qquad \dfrac{}{\varepsilon[\mathsf{y} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{y} : \hat{\rho}_0 \triangleright \mathbf{0}}}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{m}}\}\!\}, \mathsf{y} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{f}\,\mathsf{y} : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \mathbf{0}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}_{\mathsf{m}}\}\!\}, \mathsf{y} \mapsto \{\!\{\hat{\rho}_0\}\!\}] \vdash_1 \mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : (\emptyset, \{\!\{0,1,1\}\!\}, o) \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}}\,(@)}{\varepsilon[\mathsf{f} \mapsto \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}_{\mathsf{m}}\}\!\}] \vdash_1 \lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}) : \hat{\tau}_{\mathsf{m}} \triangleright \{(\mathsf{a},1),(\mathsf{b},0)\}}\,(\lambda)$$

Below the lower @ rule the information about a $(1,\mathsf{a})$-flag (from the first premiss) meets the information about a marker of order 1 (from the second premiss), and thus a $(2,\mathsf{a})$-flag is placed, which increases the flag counter.

Denote $\hat{\rho}_m = (\emptyset, M_m^{all}, o)$, where $M_m^{all} \in \mathcal{M}_m$ is such that $M_m^{all}(0) = 1$ and $M_m^{all}(k) = |A|$ for all $k \in \{1,\ldots,m\}$. The key property of the type system is described by the following theorem.

▶ **Theorem 8.** *Let $m \in \mathbb{N}$, and let P be a closed word-recognizing λ-term of sort o and complexity at most $m+1$. Then $Diag_A(\mathcal{L}(BT(P)))$ holds if and only if for every $n \in \mathbb{N}$ we can derive $\varepsilon \vdash_m P : \hat{\rho}_m \triangleright c_n$ with some c_n such that $c_n(a) \geq n$ for all $a \in A$.*

We omit the proof of this theorem. The overall idea is to follow the intuitions described on page 4, and consider only such sequences of β-reductions in which reductions of higher orders are performed before β-reductions of lower orders. The details are tedious, but rather standard. Actually, a quite similar proof was performed in our recent work [19] concerning the single-letter case.

▶ **Example 9.** Denote $\hat{\sigma}_R = (\emptyset, \{\!\{0\}\!\}, \{\!\{\hat{\tau}_{\mathsf{a}}, \hat{\tau}_{\mathsf{b}}, \hat{\tau}_{\mathsf{m}}\}\!\}{\to}o)$, where $\hat{\tau}_{\mathsf{b}} = (\{(1,\mathsf{b})\}, \mathbf{0}, \{\!\{\hat{\rho}_0\}\!\}{\to}o)$. We can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \mathbf{0}$ by descending to the first child of the outermost $\mathsf{nd}\langle\cdot,\cdot\rangle$ in $R_1 = \lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,(\mathsf{c}\langle\rangle), R_1\,(\lambda\mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle$. Then, basing on a type judgment $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \{(\mathsf{a}, c(\mathsf{a})+1), (\mathsf{b}, c(\mathsf{b}))\}$ using in particular the derivation fragment from Example 7, and similarly $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright \{(\mathsf{a}, c(\mathsf{a})), (\mathsf{b}, c(\mathsf{b})+1)\}$. By composing the above derivation fragments, we can derive $\varepsilon \vdash_1 R_1 : \hat{\sigma}_R \triangleright c$ for c that is arbitrarily large on both coordinates. Examples 4-6 contain derivations of type triples $\hat{\tau}_{\mathsf{a}}$ and $\hat{\tau}_{\mathsf{m}}$ for the λ-term $\lambda\mathsf{x}.\mathsf{nd}\langle\mathsf{a}\langle\mathsf{x}\rangle, \mathsf{b}\langle\mathsf{x}\rangle\rangle$; similarly we can derive the type triple $\hat{\tau}_{\mathsf{b}}$. Using the @ rule one more time, we can derive $\varepsilon \vdash_1 \Lambda(\mathcal{G}_1) : \hat{\rho}_1 \triangleright c$ for c that is arbitrarily large on both coordinates.

▶ **Example 10.** Consider the scheme $\mathcal{G}_2$ obtained from $\mathcal{G}_1$ by changing the rule $\mathcal{R}(\mathsf{N})$ to $\lambda\mathsf{f}.\mathsf{nd}\langle\mathsf{f}\,(\mathsf{c}\langle\rangle), \mathsf{N}\,(\lambda\mathsf{y}.\mathsf{f}\,\mathsf{y})\rangle$. Then while deriving a type for $\lambda\mathsf{y}.\mathsf{f}\,\mathsf{y}$ we can use only one type triple: either $\hat{\tau}_{\mathsf{a}}$, or $\hat{\tau}_{\mathsf{b}}$, or $\hat{\tau}_{\mathsf{m}}$, which causes that the flag counter is not increased. Thus, by adopting the derivation fragment considered in the previous example, out of $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright c$ we can only derive $\varepsilon \vdash_1 R_2 : \hat{\sigma}_R \triangleright c$, with the same flag counter (where R_2 is defined analogously to R_1). Altogether, we can derive $\varepsilon \vdash_1 \Lambda(\mathcal{G}_2) : \hat{\rho}_1 \triangleright c$ only for c with $c(\mathsf{a}) + c(\mathsf{b}) \leq 1$. This corresponds to the fact that $\mathcal{L}(\mathcal{G}_2)$ contains only words with at most one letter from $\{\mathsf{a},\mathsf{b}\}$.

▶ **Example 11.** In the derivation from Example 9 both order-1 markers were placed in the same leaf, corresponding to the subterm x. Consider, however, a scheme $\mathcal{G}_3$, where additionally to M and N we have a nonterminal $\mathsf{M_b}$ of sort o, and the rules are changed to:

$$\mathcal{R}(\mathsf{M}) = \mathsf{N}\,(\lambda \mathsf{x}.\mathsf{a}\langle \mathsf{x}\rangle)\,, \qquad \mathcal{R}(\mathsf{M_b}) = \mathsf{nd}\langle \mathsf{c}\langle\rangle, \mathsf{b}\langle \mathsf{M_b}\rangle\rangle\,,$$
$$\mathcal{R}(\mathsf{N}) = \lambda \mathsf{f}.\mathsf{nd}\langle \mathsf{f}\,\mathsf{M_b}, \mathsf{N}\,(\lambda \mathsf{y}.\mathsf{f}\,(\mathsf{f}\,\mathsf{y}))\rangle\,.$$

Here we need to place one order-1 marker (responsible for counting appearances of the a symbol) in a leaf corresponding to x, and another order-1 marker (responsible for counting appearances of the b symbol) in a leaf corresponding to $\mathsf{c}\langle\rangle$.

Effectiveness. We now justify how Theorem 3 follows from Theorem 8, i.e., how given a word-recognizing scheme $\mathcal{G}$ of order at most $m+1$ and a set of symbols A, one can check in m-EXPTIME whether $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ can be derived for flag counters c that are arbitrarily large on every coordinate. Let us say that two type judgments are equivalent if they differ only in values of the flag counter. One can see that if c is required to be large enough on every coordinate, then in the derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$, for every symbol $a \in A$, there are two equivalent type judgments lying on the same path (the path may depend on a) and such that the a-coordinate of their flag counter differs. A derivation having this property will be called *pumpable*. This name is justified, because the opposite implication also holds: if we have a pumpable type derivation, then we can repeat (as many times as we want) its fragment between all these pairs of equivalent type judgments, increasing arbitrarily all coordinates of the flag counter in the resulting type judgment. This holds thanks to the following additivity property of our type system: if out of $\Gamma \vdash_m P : \hat{\tau} \triangleright c$ we can derive $\Gamma' \vdash_m P' : \hat{\tau}' \triangleright c'$, then out of $\Gamma \vdash_m P : \hat{\tau} \triangleright d$ we can derive $\Gamma' \vdash_m P' : \hat{\tau}' \triangleright c' + d - c$.

We exploit the above equivalence, and we also observe that while deriving $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ we may only use type judgments $\Gamma \vdash_m Q : \hat{\tau} \triangleright d$ (call them *useful*) in which Q is a subterm of $\Lambda(\mathcal{G})$ and $\Gamma(x) \neq \mathbf{0}$ only for variables x that are free in Q. It is not difficult to give an algorithm that checks whether there is a pumpable derivation of $\varepsilon \vdash_m \Lambda(\mathcal{G}) : \hat{\rho}_m \triangleright c$ (for some c), and works in time polynomial in the number of equivalence classes of useful type judgments (and exponential in $|A|$).

It remains to bound the number of these equivalence classes. We first bound the number of type triples. Essentially, type triples in $\mathcal{TT}^{\alpha}_{k}$ with $\alpha = \alpha_1 \to \ldots \to \alpha_s \to o$ store "sets" of type triples from $\mathcal{TT}^{\alpha_i}_{ord(\alpha_i)}$, and thus their number grows exponentially when we increase the order of α by one. There is a slight exception for α of order 1: the number of type triples in $\mathcal{TT}^{\alpha}_{k}$ for such α is polynomial, not exponential. The reason is that, for a type $(C_1 \to \ldots \to C_s \to o) \in \mathcal{T}^{\alpha}_{k}$ with $ord(\alpha) = 1$, the triple containers $C_1, \ldots, C_s$ can contain altogether only at most one type triple (by definition it is required that $\sum_{i=1}^{s} \mathsf{Mk}(C_i)(0) \leq 1$ while, on the other hand, for type triples $\hat{\sigma} \in \mathcal{TT}^{o}_{0}$ it is required that $\mathsf{Mk}(\hat{\sigma})(0) = 1$). In effect, $|\mathcal{TT}^{\alpha}_{m}|$ for $ord(\alpha) \leq m+1$ is m-fold exponential in the size of $\mathcal{G}$. It easily follows that the number of equivalence classes of useful type judgments is also m-fold exponential in the size of $\mathcal{G}$, because all variables appearing in $\Lambda(\mathcal{G})$ are of order strictly smaller than $m+1$.

A trivial reduction from the problem of emptiness of $\mathcal{L}(\mathcal{G})$ shows that our problem is indeed m-EXPTIME-hard [15]. For $m \in \{-1, 0\}$ one can prove NP-completeness.

4 Extensions

Downward Closure. The downward closure of a language of words L, denoted $L{\downarrow}$, is the set of all scattered subwords (subsequences) of words from L. Recall that the downward closure

of any set is always a regular language; moreover, it is a finite union of *ideals*, i.e., languages of the form $Y_0^*\{x_1,\varepsilon\}Y_1^*\dots\{x_n,\varepsilon\}Y_n^*$, where $x_1,\dots,x_n$ are letters, and $Y_0,\dots,Y_n$ are sets of letters. The main interest on the diagonal problem comes from the fact that this problem is closely related to computability of the downward closure of a language of words (where we aim in presenting the results by a list of ideals, or by a finite automaton). Indeed, having a word-recognizing scheme $\mathcal{G}$, it is not difficult to compute $\mathcal{L}(\mathcal{G})\downarrow$ by performing multiple calls to a procedure solving the diagonal problem (for products of $\mathcal{G}$ and some finite automata). The complexity of this algorithm is directly related to the size of its output. We, however, do not know any upper bound on the size of (a representation of) $\mathcal{L}(\mathcal{G})\downarrow$. A recently developed pumping lemma for nondeterministic schemes [2] may shed some new light on this subject (while pumping lemmata for deterministic schemes [10, 12] seem irrelevant here).

Instead of actually computing the downward closure, Zetzsche [24] proposed to consider the following decision problem of downward-closure inclusion: given two word-recognizing schemes $\mathcal{G},\mathcal{H}$ of order at most m, check whether $\mathcal{L}(\mathcal{G})\downarrow \subseteq \mathcal{L}(\mathcal{H})\downarrow$; he proved that this problem is co-m-NEXPTIME-hard. It would be interesting to give some upper bound on the complexity of this problem. Although, again, we do know how to do this, we can at least give a partial result.

▶ **Theorem 12.** *Let $m \geq 1$. Given a word-recognizing scheme $\mathcal{H}$ of order at most $m+1$, and an ideal I, the problem of deciding whether $I \subseteq \mathcal{L}(\mathcal{H})\downarrow$ is m-EXPTIME-complete.*

This is an easy consequence of Theorem 3: it is enough to appropriately combine $\mathcal{H}$ with I, and then solve the diagonal problem.

Tree-Generating Schemes. Although the main interest on the diagonal problem is for word-recognizing schemes, the problem can be also considered for tree-recognizing schemes. Let us see how our methods can be adopted to this more general case. Consider a tree $T \in \mathcal{L}(\mathcal{G})$, and a term P_0 such that $\Lambda(\mathcal{G}) \to_\beta^* P_0$ and $P_0 \to_{\mathsf{nd}}^* T$, i.e., that T can be found in a prefix of P_0. In the word case, we were placing order-1 flags in node constructors of T, and then we continued using the fact that they are all aligned along one path (as actually T consisted of a single path). This is no longer possible in the tree case. In order to resolve this issue, we additionally use flags of order 0, and we place them in node constructors of T (dispersed on multiple paths). Then, we choose only $|A|$ paths, by placing order-1 markers in $|A|$ leaves of P_0, and for every node labeled by a $(0,a)$-flag we place a $(1,a)$-flag in the closest ancestor that lies on a chosen path. In effect all order-1 flags are concentrated on only $|A|$ paths, and we can continue as in the word case. The described modification causes an exponential growth of the number of types, which results in the following theorem.

▶ **Theorem 13.** *For $m \geq 1$, the diagonal problem for tree-recognizing order-m schemes is m-EXPTIME-complete. For $m = 0$ it is NP-complete.*

Downward Closure for Trees. One can also consider the downward closure of a language of trees, defined as a set of all trees that can be homeomorphically embedded in trees from the language. By Kruskal's tree theorem [17] downward closures of tree languages are regular languages of trees. We notice, however, that (unlike for words) an algorithm solving the diagonal problem is highly insufficient for the purpose of computing the downward closure. Even in the single-letter case, in order to compute $L\downarrow$, one has to check, in particular, whether for every $n \in \mathbb{N}$, a full binary tree of depth n can be embedded in some tree from L; using the diagonal problem, we can only determine whether L contains arbitrarily large trees. Extending our techniques to this kind of problems is an interesting direction for further work.

References

1 Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.

2 Kazuyuki Asada and Naoki Kobayashi. Pumping lemma for higher-order languages. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 97:1–97:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.97.

3 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

4 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

5 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.

6 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.

7 Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In Adrian Kosowski and Igor Walukiewicz, editors, *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, volume 9210 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2015. doi:10.1007/978-3-319-22177-9_14.

8 Werner Damm. The IO- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

9 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

10 Alexander Kartzow and Pawel Parys. Strictness of the collapsible pushdown hierarchy. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2012. doi:10.1007/978-3-642-32589-2_50.

11 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 416–428. ACM, 2009. doi:10.1145/1480881.1480933.

12 Naoki Kobayashi. Pumping by typing. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 398–407. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.46.

13 Naoki Kobayashi, Kazuhiro Inaba, and Takeshi Tsukada. Unsafe order-2 tree languages are context-sensitive. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2014. doi:10.1007/978-3-642-54830-7_10.

14 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

15 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011. doi:10.2168/LMCS-7(4:9)2011.

16 Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015. doi:10.1016/j.ic.2014.12.015.

17 Joseph. B. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960. doi:10.2307/1993287.

18 Pawel Parys. A characterization of lambda-terms transforming numerals. *J. Funct. Program.*, 26:e12, 2016. doi:10.1017/S0956796816000113.

19 Pawel Parys. Intersection types and counting. In Naoki Kobayashi, editor, *Proceedings Eighth Workshop on Intersection Types and Related Systems, ITRS 2016, Porto, Portugal, 26th June 2016.*, volume 242 of *EPTCS*, pages 48–63, 2016. doi:10.4204/EPTCS.242.6.

20 Paweł Parys. Homogeneity without loss of generality. Submitted, 2017.

21 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

22 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

23 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *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 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.

24 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.123.

A Combinatorial Proof of Ihara-Bass's Formula for the Zeta Function of Regular Graphs

Bharatram Rangarajan

School of Computer Science, Tel Aviv University, Israel
bharatram.rangarajan@gmail.com

Abstract

We give an elementary combinatorial proof of Bass's determinant formula for the zeta function of a finite regular graph. This is done by expressing the number of non-backtracking cycles of a given length in terms of Chebyshev polynomials in the eigenvalues of the adjacency operator of the graph. A related observation of independent interest is that the Ramanujan property of a regular graph is equivalent to tight bounds on the number of non-backtracking cycles of every length.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases non-backtracking, Ihara zeta, Chebyshev polynomial, Ramanujan graph, Hashimoto matrix

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.46

1 Introduction

1.1 Ramanujan graphs and Non-backtracking Cycles

For a fixed $d \geq 3$, a family G_n of d-regular n-vertex connected graphs is said to be an *expander* family if the second largest eigenvalues of the corresponding adjacency matrices are uniformly bounded away from d. It is easy to show (using a simple application of the probabilistic method) that a random d-regular graph family is an expander family with high probability. The question as to how small the second largest eigenvalue can get is answered by the Alon-Bopanna bound [16]: For fixed $d \geq 3$, the second largest eigenvalue, in absolute value, is at least $2\sqrt{d-1} - o_n(1)$. The occurrence of the term $2\sqrt{d-1}$ in this setting is related to it being the spectral radius of (the adjacency operator of) the *universal cover* of a d-regular graph (which is the infinite d-regular tree).

▶ **Definition 1.** For $d \geq 3$, a finite connected d-regular graph G is said to be *Ramanujan* if every eigenvalue $\mu \in \mathbb{R}$ of the adjacency matrix A of G with $|\mu| \neq d$ satisfies

$$|\mu| \leq 2\sqrt{d-1}$$

In other words, a family of Ramanujan graphs is the "optimal" expander family in light of the Alon-Bopanna lower bound. Ramanujan graphs were defined and explicitly constructed by Lubotzky, Phillips and Sarnak [11] for $d-1$ being a prime, and extended by Morgenstern [14] for $d-1$ being a prime power. Their constructions used deep results from modern number theory (in particular a conjecture of Ramanujan which was later settled by Deligne et al). However, the existence (leave alone explicit constructions) of Ramanujan graph families for general $d \geq 3$ remained open for a long time until Marcus, Spielman and Srivastava [13] used the method of interlacing polynomials to establish the existence of bipartite Ramanujan families for every $d \geq 3$. For a broad survey of Ramanujan graphs, expander families and

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 46; pp. 46:1–46:13

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

their applications, the reader is referred to Murty's monograph [15] and the survey by Hoory, Linial and Wigderson [6].

The Ramanujan property (or more generally, the spectrum of the adjacency matrix) of a graph is reflected in the error term in the number of closed walks on the graph of a given length. More precisely, suppose $d = \mu_0 \geq \mu_1 \geq \cdots \geq \mu_{n-1} \geq -d$ are the n eigenvalues of an n-vertex d-regular graph. Then we know that for every $k \geq 1$, $Tr(A^k) = d^k + \sum_{j=1}^{n-1} \mu_j^k$ is precisely the number of closed walks of length k on G. We can express this succinctly using Jacobi's identity:

$$\frac{1}{det(I - At)} = \prod_{j=0}^{n-1} \frac{1}{1 - \mu_j t} = exp\left(\sum_{k=1}^{\infty} Tr(A^k)\frac{t^k}{k}\right)$$

Now if $|\mu_j| \leq 2\sqrt{d-1}$ for every $1 \leq j \leq n-1$, then the number of closed walks of length k in G is d^k (or roughly a $1/n$ fraction of the total number nd^k of walks of length k), and with a small error term of $O(n2^k d^{k/2})$.
As we shall soon see in more detail, a stronger connection between the Ramanujan property and closed walks on the graph reveals itself when we restrict the closed walks to be *non-backtracking*. An instance of backtracking is said to occur in a walk when a traversed edge is followed immediately by its reversal. Let N_k denote the number of non-backtracking cycles of length k on G. We can construct an analogous generating function for $\{N_k\}_{k\geq 1}$ as:

$$exp\left(\sum_{k=1}^{\infty} N_k \frac{t^k}{k}\right)$$

As it so happens, the above formal series can also be expressed as the inverse of a polynomial $det(I - Ht)$ where H is the Hashimoto non-backtracking walk matrix of G, which is the adjacency matrix of the oriented line digraph of G. This formal series is called the *Ihara zeta function* of the graph, denoted $\zeta_G(t)$.

1.2 Zeta functions and Riemann Hypotheses

In the 158 years since Bernard Riemann published his seminal work "On the Number of Primes Less Than a Given Magnitude", there have been several generalizations of the Riemann zeta function in various settings. Broadly speaking, a zeta function is a complex function which when expressed as an appropriate series, yields a coefficient sequence that counts "objects" of a given "weight" assembled from an underlying set of building blocks or "primes". For instance, the Riemann zeta function corresponds to a Dirichlet series where the coefficient of $1/k^s$ counts the number of positive integers (constructed using the primes of $\mathbb{Z}$ as building blocks) of absolute value k (which in this case is trivially 1 for every $k \in \mathbb{N}$). The utility of a zeta function arises from the fact that many interesting properties of the underlying structure can be inferred from the zeros and poles of the corresponding zeta function.

The precursor to the zeta function of a graph, as we know it today, is the Selberg zeta function of a Riemannian manifold. For a hyperbolic surface $M = \Gamma/H$, the Selberg zeta function $\gamma_M(s)$ is an Euler product over the set of all primitive closed geodesics of M. The zeros and poles of the Selberg zeta function appear in the Selberg trace formula, which relates the distribution of primes with the spectrum of the Laplace-Beltrami operator of the surface. This line of study was further extended by Ihara [7] to obtain a p-adic analogue of

the Selberg trace formula, opening up further avenues for the study of geodesic zeta functions in discrete settings. The idea of considering closed geodesics as primes inspired the work of Hashimoto [5], Hyman Bass [2], Kotani and Sunada [9] to study the analogous notion in the discrete setting of a finite graph, using the prime cycle classes of the graph in place of primitive geodesics. We shall construct the zeta function of a graph in this way in section 2. Just like the Selberg zeta function is related to the spectrum of the Laplace-Beltrami operator of the surface, it is natural to ask if its discrete analogue, the Ihara zeta function of a graph, is related to the spectrum of the Laplacian matrix of the graph. This is precisely the result of Bass [2] who gives an elegant expression for the Ihara zeta function of a graph $G = (V, E)$ as follows:

▶ **Theorem 2** (Bass). *For a finite connected graph $G = (V, E)$,*

$$\zeta_G(t) = \frac{1}{(1-t^2)^{|E|-|V|} det(I - tA + (D-I)t^2)}$$

where A is the adjacency matrix of G and D is the diagonal matrix of degrees of the vertices of G, or in other words, $D = diag(A\vec{1})$.

In particular, if G is a d-regular connected graph, then

$$\zeta_G(t) = \frac{1}{(1-t^2)^{|E|-|V|} det(I - tA + (d-1)t^2)}$$

This immediately tells us what the poles of the Ihara zeta function are. The significance of these poles arises from a surprising analogue of the classical Riemann hypothesis in our present context. The classical Riemann hypothesis for the Riemann zeta function $\zeta(t)$ states that every non-trivial zero of $\zeta(t)$ lies on the line $Re(z) = 1/2$ in the complex plane. Analogues of the Riemann hypothesis can be formulated for other zeta functions too. For instance, the Riemann hypothesis for curves over finite fields states that every zero of the Hasse-Weil zeta function for a projective curve over a finite field $\mathbb{F}_q$ is of absolute value exactly $1/\sqrt{q}$. It is interesting to note that while the classical Riemann hypothesis remains elusive, the Riemann hypothesis for finite fields has been proved, and is one of the crowning achievements of twentieth-century mathematics.

It is natural to ask if there is an appropriate formulation of a Riemann hypothesis for the Ihara zeta function, and what it means for the graph. Recall that a d-regular graph G is *Ramanujan* if for every eigenvalue $\mu \in \mathbb{R}$ of the adjacency matrix of G with $|\mu| \neq d$ satisfies $|\mu| \leq 2\sqrt{d-1}$. Combining this with Bass's determinant formula for the zeta function of G, if can be easily shown [15] that

▶ **Lemma 3** (Riemann Hypothesis for graphs). *A d-regular graph G is Ramanujan iff every pole $\lambda \in \mathbb{C}$ of $\zeta_G(t)$ such that $|\lambda| \neq 1$ and $|\lambda| \neq (d-1)^{-1}$ satisfies*

$$|\lambda| = \frac{1}{\sqrt{d-1}}$$

1.3 Proof Sketch

There exist several proofs [9] [17] of theorem 2, and most proofs start by expressing the zeta function in terms of not the adjacency matrix A of G, but the adjacency matrix H of the oriented line digraph of G (the Hashimoto non-backtracking walk matrix). After all, $\zeta_G(t)^{-1} = det(I - Ht)$, and so the problem reduces to expressing $det(I - Ht)$ in terms of

the adjacency matrix A. We shall briefly sketch the standard proof in the next section once we have the preliminaries in place. There also exists another purely combinatorial proof by Foata and Zeilberger [4] employing the algebra of Lyndon words.

In this paper, we shall see an elementary proof of theorem 2 for the special case when G is d-regular. While the assumption of regularity is certainly a limitation, it allows for a more transparent combinatorial proof. The basic idea is outlined as follows:

- We use the fact that the zeta function $\zeta_G(t)$ has an expansion of the form

$$\zeta_G(t) = exp\left(\sum_{k=1}^{\infty} N_k \frac{t^k}{k}\right)$$

  where for $k \in \mathbb{N}$, N_k is the number of non-backtracking cycles in G of length k. This is explored in section 2.
- While an expression for N_k is not immediate, a natural starting point is the study of non-backtracking walks on G. We can construct the family $\{A_k\}_{k\in\mathbb{Z}_{\geq 0}}$ of $n \times n$ matrices such that for every $k \in \mathbb{Z}_{\geq 0}$ and every $v, w \in V$, $(A_k)_{v,w}$ is the number of non-backtracking walks on G of length k from v to w. We shall discuss the construction of these non-backtracking walk matrices in section 3.
- While it might be tempting to claim that $N_k = Tr(A_k)$, unfortunately that is not the case! However, while they may not be equal, they are indeed precisely related. In section 4, we develop a combinatorial lemma to relate N_k and $Tr(A_k)$. This expression, while simple, could prove useful and is of independent interest.
- The combinatorial lemma greatly simplifies the problem since $Tr(A_k)$ is well-understood in terms of the eigenvalues of A and a family of orthogonal polynomials called the Chebyshev polynomials. We shall put these ingredients together in section 5 to arrive at Bass's determinant formula.

Essentially, the main contribution of this paper is a proof of following lemma:

▶ **Lemma 4.** *Let G be a finite, connected d-regular graph on n vertices, and suppose $d = \mu_0 \geq \mu_1 \geq \cdots \geq \mu_{n-1} \geq -d$ are the n real eigenvalues of its adjacency matrix A. Let N_k be the number of non-backtracking cycles of length k on G. Then*

$$N_k = \begin{cases} \sum_{j=0}^{n-1} 2(d-1)^{k/2} T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right) & \text{if } k \text{ is odd} \\ n(d-2) + \sum_{j=0}^{n-1} 2(d-1)^{k/2} T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right) & \text{if } k \text{ is even} \end{cases}$$

where T_k is the k-th Chebyshev polynomial of the first kind.

While the above expression is easy to derive given Bass's determinant formula for the zeta function, our proof proceeds in the other direction: by establishing this expression first and then using it to derive Bass's determinant formula using the generating function for Chebyshev polynomials of the first kind.

An interesting consequence of the above formula for N_k is an interpretation of the summand corresponding to the trivial eigenvalue d of G. Using a standard explicit formula for the polynomial T_k given by

$$T_k(x) = \begin{cases} \cos(k \arccos x) & \text{if } |x| \leq 1 \\ \frac{1}{2}(x - \sqrt{x^2-1})^k + \frac{1}{2}(x + \sqrt{x^2-1})^k & \text{if } |x| > 1 \end{cases}$$

we can compute $T_k(d/2\sqrt{d-1})$ to get

$$T_k\left(\frac{d}{2\sqrt{d-1}}\right) = (d-1)^k + 1$$

So if G is non-bipartite, we get

$$N_k = \begin{cases} (d-1)^k + 1 + \sum_{j=1}^{n-1} 2(d-1)^{k/2} T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right) & \text{if k is odd} \\ (d-1)^k + 1 + (d-2) + \sum_{j=1}^{n-1} \left((d-2) + 2(d-1)^{k/2} T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right)\right) & \text{if k is even} \end{cases}$$

In particular, when G is Ramanujan,

$$N_k = \begin{cases} (d-1)^k + 1 + O(nd^{k/2}) & \text{if k is odd} \\ (d-1)^k + 1 + (d-2) + O(nd^{k/2}) & \text{if k is even} \end{cases}$$

It is known that T_k is an odd function when k is odd, and an even function when k is even. So when G is bipartite,

$$N_k = \begin{cases} 0 & \text{if k is odd} \\ 2(d-1)^k + 2 + 2(d-2) + \sum_{j=1}^{n-2} \left((d-2) + 2(d-1)^{k/2} T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right)\right) & \text{if k is even} \end{cases}$$

and in particular when G is a bipartite Ramanujan graph,

$$N_k = \begin{cases} 0 & \text{if k is odd} \\ 2(d-1)^k + 2 + 2(d-2) + O(nd^{k/2}) & \text{if k is even} \end{cases}$$

It is interesting to ask what these dominant terms represent. It is known [12] that the number of *cyclically reduced words* of length k in a free group of rank m is exactly $(2m-1)^k + 1$ when k is odd, and $(2m-1)^k + 2m - 1$ when k is even. So if G were a Cayley graph of a group Γ and a symmetric generating set S (without involutive elements) of size d, then consider the walks on G corresponding to a choice of root and a cyclically reduced word over S of length k. The total number of such walks is $n\left((d-1)^k + 1\right)$ if k is odd, and $n\left((d-1)^k + 1 + (d-2)\right)$ if k is odd. If G is non-bipartite, we would expect a $1/n$ fraction of these walks to return to the root (that is, become non-backtracking cycles). If G is bipartite, then for even k, we would expect a $2/n$ fraction of these walks to be non-backtracking cycles (as there are now only $n/2$ candidates for the end vertex). These quantities are precisely the ones that appear as the dominant terms in the expressions for N_k.
So the Ramanujan property (or the graph Riemann hypothesis) implies that the number N_k of non-backtracking cycles is close to the expected value, with optimally tight error term.

2 Preliminaries

2.1 Non-backtracking cycles and the Ihara Zeta Function

For an integer $d \geq 2$, let $G = (V, E)$ be a finite d-regular undirected graph with adjacency matrix A. A *walk* on the graph G is a sequence $v_0 v_1 \ldots v_k$ where $v_0, v_1, \ldots, v_k$ are (not necessarily distinct) vertices in V, and for every $0 \leq i \leq k-1$, $(v_i, v_{i+1}) \in E$. The vertex v_0 is referred to as the *root* (or origin) of the above walk, v_k is the terminus of the walk, and the walk is said to have length k.

It is often useful to equivalently define a walk as a sequence of directed or oriented edges. Associate each edge $e = (v, w) \in E$ with two directed edges (or rays) denoted

$$\vec{e} = (v \to w) \text{ and } \vec{e}^{-1} = (w \to v)$$

Note that the origin $org(\vec{e})$ is the vertex v and its terminus $ter(\vec{e})$ is the vertex w. Similarly, the origin $org(\vec{e}^{-1})$ is the vertex w and its terminus $ter(\vec{e})$ is the vertex v. Let $\vec{E}$ denote the set of $m = nd$ directed edges of G. So a walk of length k can equivalently be described as a sequence $\vec{e}_1\vec{e}_2 \dots \vec{e}_k$ of k (not necessarily distinct) oriented edges in $\vec{E}$ such that for every $1 \leq i \leq k-1$, $ter(\vec{e}_i) = org(\vec{e}_{i+1})$. This is a walk that starts at $org(\vec{e}_1)$ and ends at $ter(\vec{e}_k)$. It is easy to show that for any $k \in \mathbb{N}$, the number of walks of length k between vertices $u, v \in V$ is exactly $(A^k)_{u,v}$. In particular, the total number of closed walks of length k in G is exactly $Tr(A^k)$.

▶ **Definition 5.** A *non-backtracking walk* of length k from $v_0 \in V$ to $v_k \in V$ is a walk $v_0v_1 \dots v_k$ such that for every $1 \leq i \leq k-1$, $v_{i-1} \neq v_{i+1}$. Equivalently, a non-backtracking walk of length k from $v \in V$ to $w \in V$ is a walk $\vec{e}_1\vec{e}_2 \dots \vec{e}_k$ such that $org(\vec{e}_1) = v$, $ter(\vec{e}_k) = w$ and for every $1 \leq i \leq k-1$, $\vec{e}_{k+1} \neq \vec{e}_k^{-1}$.

▶ **Definition 6.** A *non-backtracking cycle* of length k with root v is a non-backtracking closed walk $v, v_1, v_2, \dots, v_{k-1}, v$ with the additional boundary constraint that $v_1 \neq v_{k-1}$.

Non-backtracking random walks (NBRW) on graphs have been studied in the context of mixing time [1], cut-offs [10], and exhibit more useful statistical properties than simple random walks (SRW).

Let $\mathcal{C}$ denote the set of all non-backtracking cycles in G, and for $C \in \mathcal{C}$, let $|C|$ denote the length of the cycle C. There are two elementary constructions we can carry out to generate more elements of $\mathcal{C}$ from a given cycle C:

- *Powering*: Given a non-backtracking cycle $C \in \mathcal{C}$ of length k of the form $C = \vec{e}_1\vec{e}_2 \dots \vec{e}_k$ and $m \geq 1$, define the power

  $$C^m = \underbrace{\vec{e}_1 \dots \vec{e}_k . \vec{e}_1 \dots \vec{e}_k \dots \vec{e}_1 \dots \vec{e}_k}_{\text{m times}}$$

  which is the concatenation of the string of edges corresponding to the walk C with itself m times. Note that C^m is also a non-backtracking cycle in G of length mk. Essentially, C^m represents the walk obtained by repeating or winding the walk C m times. Also note that C and C^m are both rooted at the same vertex. A cycle $P \in \mathcal{C}$ shall be called a *prime* cycle if there exists no element $C \in \mathcal{C}$ and $m \geq 2$ such that $P = C^m$. Essentially, a prime cycle in $\mathcal{C}$ is one that is not a repeated winding of a simpler cycle in $\mathcal{C}$. Note that every element of $\mathcal{C}$ is either a prime or a prime power.
- *Cycle Equivalence*: Given a non-backtracking cycle $C \in \mathcal{C}$ of length k of the form $C = \vec{e}_1\vec{e}_2 \dots \vec{e}_k$, we can form another walk $C^{(2)} = \vec{e}_2\vec{e}_3 \dots \vec{e}_k\vec{e}_1$ which is also a non-backtracking cycle in G of length k, but now rooted at the origin of the directed edge $\vec{e}_2$ (or the terminus of $\vec{e}_1$). More generally, for $1 \leq j \leq k$, define

  $$C^{(j)} = \vec{e}_j\vec{e}_{j+1} \dots \vec{e}_k\vec{e}_1\vec{e}_2 \dots \vec{e}_{j-1}$$

  which is a cyclic permutation of the walk C obtained by choosing a different root. So given a cycle $C \in \mathcal{C}$ of length k, we get $k-1$ additional cycles in $\mathcal{C}$ of length k for free this way. In fact, this defines an equivalence class $\sim$ on $\mathcal{C}$, and the set $[C] = \{C^{(1)}, C^{(2)}, \dots, C^{(k)}\}$ is called the equivalence class of C. An element $[C] \in \mathcal{C}/\sim$ represents a non-backtracking cycle modulo a choice of root.

We can now formally define the zeta function of the graph G. For simplicity, let us assume that G is connected and does not have any leaves (or vertices of degree 1).

▶ **Definition 7.** Let $\mathcal{P}$ denote the set of equivalence classes of prime non-backtracking cycles in G. The Euler product

$$\prod_{[P]\in\mathcal{P}} \frac{1}{1-t^{|P|}}$$

is called the *Ihara zeta function* of the graph G, denoted $\zeta_G(t)$.

Let N_k denote the number of non-backtracking cycles in G of length k. Then observe that

$$\sum_{k=1}^{\infty} N_k \frac{t^k}{k} = \sum_{\text{prime } P} \frac{1}{|P|}\left(\sum_{m=1}^{\infty} \frac{t^{m|P|}}{m}\right) = -\sum_{[P]\in\mathcal{P}} \log\left(1-t^{|P|}\right)$$

Thus,

$$\zeta_G(t) = \prod_{[P]\in\mathcal{P}} \frac{1}{1-t^{|P|}} = exp\left(\sum_{k=1}^{\infty} N_k \frac{t^k}{k}\right)$$

Just like the number of cycles in G of length k is $Tr(A^k)$, we can describe the number N_k of non-backtracking cycles in G of length k as the trace of the matrix H^k where H is the *Hashimoto non-backtracking walk matrix* of G defined as follows: $H \in \mathbb{C}^{dn\times dn}$ with

$$H_{i,j} = \begin{cases} 1 & \text{if } \vec{e}_j \neq \vec{e}_i^{-1} \text{ and } ter(\vec{e}_i) = org(\vec{e}_j) \\ 0 & \text{otherwise} \end{cases}$$

In other words, the entry $H_{i,j}$ is an indicator for whether the oriented edge $\vec{e}_i$ feeds into the oriented edge $\vec{e}_j$ allowing us to form a non-backtracking walk $\vec{e}_i\vec{e}_j$ of length 2. Note that unlike A, the Hashimoto matrix H is *not* a symmetric matrix, and hence it need not have all real eigenvalues and an associated orthonormal eigenbasis. The interested reader is referred to [10] where the authors work out the precise eigendecomposition of the Hashimoto matrix H.
It is clear that for every $k \in \mathcal{N}$,

$$N_k = Tr(H^k)$$

and so

$$\zeta_G(t) = exp\left(\sum_{k=1}^{\infty} Tr(H^k)\frac{t^k}{k}\right) = exp\left(-Tr\left(\log\left(I-tH\right)\right)\right)$$

By Jacobi's formula relating the trace of the logarithm of a matrix to the logarithm of its determinant, we get

$$\zeta_G(t) = \frac{1}{det(I-Ht)}$$

In particular, this establishes the rationality of the Ihara zeta function of a regular graph, and further implies that the reciprocal $\zeta_G(t)^{-1}$ is a polynomial in t over $\mathbb{Z}$ of degree at most $m = nd$. However, it is not immediate what the spectrum of H is. Thus, in a sense, Bass's determinant formula for regular graphs can be interpreted as a way to determine the

spectrum of the Hashimoto matrix H in terms of the spectrum of the adjacency matrix A. Now that we have defined the Hashimoto non-backtracking walk matrix H, we shall briefly sketch a previous proof of Bass's determinant formula (from [8]) involving expressing both the matrix H and the adjacency matrix A in terms of two directed edge incidence matrices S and T and a backtracking matrix B defined as follows: For a directed edge $(u \to v)$ and a vertex $w \in V$, define

$$S_{w,(u\to v)} = \begin{cases} 1 & \text{if } u = w \\ 0 & \text{otherwise} \end{cases} \qquad T_{(u\to v),w} = \begin{cases} 1 & \text{if } v = w \\ 0 & \text{otherwise} \end{cases}$$

Note that S is an $n \times dn$ edge incidence matrix that indicates the origin or starting vertex of the directed edge, while T is a $dn \times n$ edge incidence matrix that indicates the terminating vertex of the directed edge. It can be checked that

$$A = ST$$

$$H = TS - B$$

$$SBT = dI$$

where B is a $dn \times dn$ backtracking indicator matrix defined as

$$B_{(u\to v),(w\to z)} = \begin{cases} 1 & \text{if } v = w \text{ and } u = z \\ 0 & \text{otherwise} \end{cases}$$

Now that A and H are related through S, T and B, standard matrix manipulation would suffice to show that

$$det(I - Ht) = (1 - t^2)^{\frac{dn}{2} - n} det\left(I - At + (d-1)t^2\right)$$

In fact, this proof works even when the graph G is not regular. However, the linear-algebraic calculations, though simple, tend to mask the underlying combinatorial structure.

2.2 Non-backtracking Walks and Chebyshev Polynomials

Just like $(A^k)_{v,w}$ counts the total number of walks on G from v to w (with backtrackings) of length k, we can construct a family

$$A_0, A_1, A_2, A_3, \ldots$$

of $n \times n$ matrices over $\mathbb{C}$ such that the value $(A_k)_{v,w}$ is the number of non-backtracking walks on G from v to w of length k. This family $\{A_k\}_{k\in\mathbb{N}}$ can be inductively defined using powers of A as follows:

- $A_0 = I$ and $A_1 = A$
- $A_2 = A^2 - dI$
- For $k \geq 3$,

$$A_k = A_{k-1}A - (d-1)A_{k-2}$$

The recurrence relation above can be used to easily show that the ordinary (matrix) generating function for the above sequence, with some mild abuse of notation, is

$$\sum_{k=0}^{\infty} t^k A_k = \frac{1 - t^2}{I - At + (d-1)t^2}$$

The generating function above is closely related to the generating function of a well-studied family of orthogonal polynomials. Consider the family of Chebyshev polynomials $\{U_k\}_{k\geq 0}$ of the second kind, which are univariate complex polynomials defined by the recurrence

$$U_0(x) = 1 \text{ and } U_1(x) = 2x$$

and for $k \geq 2$,

$$U_k(x) = 2xU_{k-1}(x) - U_{k-2}(x)$$

and with generating function

$$\sum_{k=0}^{\infty} U_k(x)t^k = \frac{1}{1-2xt+t^2}$$

It can be shown [3] that for every $k \geq 2$,

$$\sum_{0\leq j\leq k/2} A_{k-2j} = (d-1)^{k/2} U_k\left(\frac{A}{2\sqrt{d-1}}\right)$$

and so by taking trace on both sides we get

$$\sum_{0\leq j\leq k/2} Tr(A_{k-2j}) = (d-1)^{k/2} \sum_{j=0}^{n-1} U_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right)$$

where

$$d = \mu_0 \geq \mu_1 \geq \cdots \geq \mu_{n-1} \geq -d$$

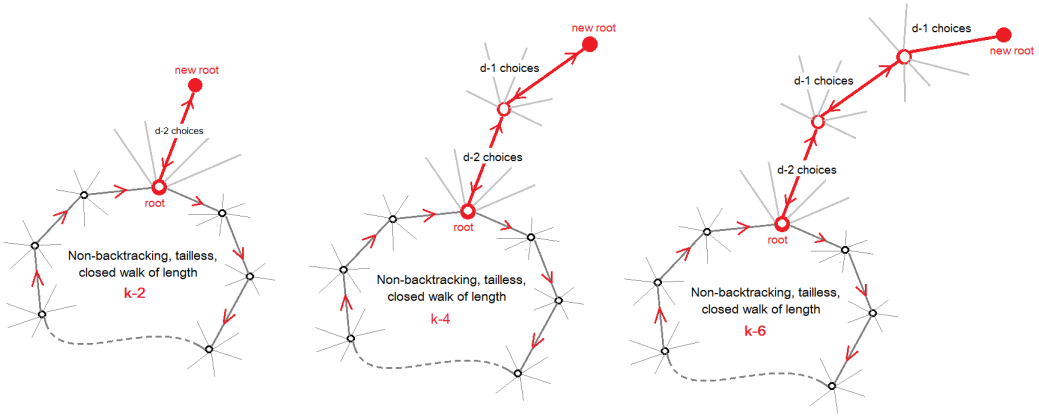
are the n eigenvalues of the adjacency matrix A. Thus we have an expression for the trace of A_k as a polynomial in the eigenvalues of A. This approach is used in the seminal work of Lubotzky, Phillips and Sarnak in their construction of Ramanujan graphs [11], and for a more detailed exposition of Chebyshev polynomials and non-backtracking walks on regular graphs, the reader is referred to the monograph by Davidoff, Sarnak and Valette [3].

While $(A_k)_{v,w}$ counts the number of walks on G from vertex v to vertex w without backtracking, observe that the diagonal element $(A_k)_{v,v}$ does *not* count the number of non-backtracking cycles of length k rooted at v. This is because $(A_k)_{v,v}$ also counts walks of the form $\vec{e}_1\vec{e}_2\ldots\vec{e}_k$ where $\vec{e}_{i+1} \neq \vec{e}_i^{-1}$ for any $1 \leq i \leq k-1$ but $\vec{e}_k = \vec{e}_1^{-1}$. That is, $\vec{e}_1\vec{e}_2\ldots\vec{e}_k$ is non-backtracking as a walk from v to v, but when considered as a closed walk, the two end edges form a backtracking and is hence not a non-backtracking cycle! Such an instance of a backtracking that gets overlooked in $Tr(A_k)$ shall be referred to as a *tail*.
So $Tr(A_k)$ counts the number of closed walks of length k that could have at most 1 tail (and hence does *not* count the non-backtracking cycles of length k). Denote $Tr(A_k)$ by M_k. In the following section, we shall establish a simple but useful combinatorial lemma relating M_k with N_k.

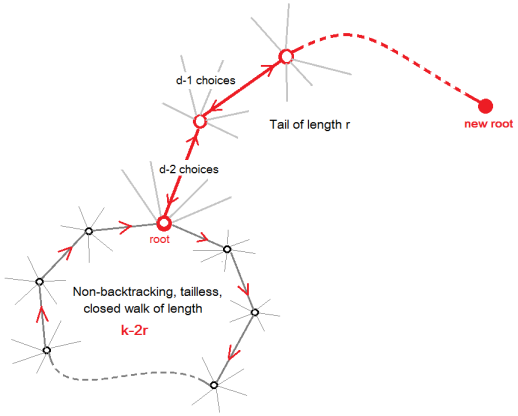
3 The Combinatorial Lemma

Firstly it is clear that $N_1 = N_2 = 0$. For $k \geq 3$, we can count the number M_k of *tailed* non-backtracking, closed walks of length k based on the length of the tail as illustrated below:

- A tailless, non-backtracking closed walk of length k, and there are N_k of them.
- A tailless, non-backtracking closed walks of length $k-2$ and a tail of length 1. Since the root is fixed and there are $d-2$ choices for the tail (and consequently, the new root), the number of non-backtracking closed walks of length k with a tail of length 1 is $(d-2)N_{k-2}$.
- A tailless, non-backtracking closed walks of length $k-4$ and a tail of length 2. In this case the first vertex of the tail can be chosen in $d-2$ ways, and the next vertex (the new root) can be chosen in $d-1$ ways. So the number of non-backtracking closed walks of length k with a tail of length 2 is $(d-1)(d-2)N_{k-4}$.



More generally, for $2 \le r \le \lfloor k/2 \rfloor$, the number of non-backtracking closed walks of length k with a tail of length r is $(d-1)^{r-1}(d-2)N_{k-2r}$.



Thus for every $k \ge 3$,

$$\begin{aligned} M_k =& N_k + (d-2)N_{k-2} + (d-2)(d-1)N_{k-4} + (d-2)(d-1)^2 N_{k-6} \\ &+ \cdots + (d-2)(d-1)^{\lfloor \frac{k-1}{2} \rfloor - 1} N_{k-2\lfloor \frac{k-1}{2} \rfloor} \end{aligned}$$

While this expression looks cumbersome, observe that

$$M_k - N_k = (d-2)\left(N_{k-2} + (d-1)N_{k-4} + \cdots + (d-1)^{\lfloor \frac{k-1}{2} \rfloor - 1} N_{k-2\lfloor \frac{k-1}{2} \rfloor} \right)$$

and a straightforward summation shows that

$$\sum_{j=1}^{\lfloor \frac{k-1}{2} \rfloor} M_{k-2j} = N_{k-2} + (d-1)N_{k-4} + \cdots + (d-1)^{\lfloor \frac{k-1}{2} \rfloor - 1} N_{k-2\lfloor \frac{k-1}{2} \rfloor}$$

▶ **Lemma 8.** *For every $k \geq 3$,*

$$N_k = \begin{cases} M_k - (d-2)(M_{k-2} + M_{k-4} + \cdots + M_1) & \text{if } k \text{ is odd} \\ M_k - (d-2)(M_{k-2} + M_{k-4} + \cdots + M_2) & \text{if } k \text{ is even} \end{cases}$$

4 The Bass Determinant Formula

From the combinatorial lemma established in the previous section and the linearity of trace, we get

$$N_k = \begin{cases} Tr\left(A_k - (d-2)(A_{k-2} + A_{k-4} + \cdots + A_1)\right) & \text{if k is odd} \\ Tr\left(A_k - (d-2)(A_{k-2} + A_{k-4} + \cdots + A_2)\right) & \text{if k is even} \end{cases}$$

Recall that

$$\sum_{0 \leq j \leq k/2} A_{k-2j} = (d-1)^{k/2} U_k \left(\frac{A}{2\sqrt{d-1}}\right)$$

So for odd k

$$\begin{aligned} & A_k - (d-2)(A_{k-2} + A_{k-4} + \cdots + A_1) \\ &= (A_k + A_{k-2} + \cdots + A_1) - (d-1)(A_{k-2} + A_{k-4} + \cdots + A_1) \\ &= (d-1)^{k/2} U_k \left(\frac{A}{2\sqrt{d-1}}\right) - (d-1)^{k/2} U_{k-2} \left(\frac{A}{2\sqrt{d-1}}\right) \end{aligned}$$

Similarly for even k,

$$\begin{aligned} & A_k - (d-2)(A_{k-2} + A_{k-4} + \cdots + A_2) \\ &= (A_k + A_{k-2} + \cdots + A_2) - (d-1)(A_{k-2} + A_{k-4} + \cdots + A_2) \\ &= (d-1)^{k/2} U_k \left(\frac{A}{2\sqrt{d-1}}\right) - (d-1)^{k/2} U_{k-2} \left(\frac{A}{2\sqrt{d-1}}\right) + (d-2)I \end{aligned}$$

As it so happens, the polynomial

$$U_k(x) - U_{k-2}(x) = 2T_k(x)$$

where $T_k(x)$ is called the *Chebyshev polynomial of the first kind* of order k. The Chebyshev polynomials of the first kind are defined in a way very similar to the Chebyshev polynomials of the second kind:

$$T_0(x) = 1$$

$$T_1(x) = x$$

and for $k \geq 2$,

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

It is easy to show that $T_k(x)$ has a generating function

$$\sum_{k=0}^{\infty} T_k(x) t^k = \frac{1 - xt}{1 - 2xt + t^2}$$

It is convenient to express N_k in terms of Chebyshev polynomials of the first kind as follows:

$$N_k = \begin{cases} Tr\left(2(d-1)^{k/2}T_k\left(\frac{A}{2\sqrt{d-1}}\right)\right) & \text{if k is odd} \\ Tr\left(2(d-1)^{k/2}T_k\left(\frac{A}{2\sqrt{d-1}}\right) + (d-2)I\right) & \text{if k is even} \end{cases}$$

This simplifies to

$$N_k = \begin{cases} \sum_{j=0}^{n-1} 2(d-1)^{k/2}T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right) & \text{if k is odd} \\ n(d-2) + \sum_{j=0}^{n-1} 2(d-1)^{k/2}T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right) & \text{if k is even} \end{cases}$$

The generating function for N_k is given by

$$\begin{aligned} \sum_{k=1}^{\infty} N_k t^k &= n(d-2)(t^2+t^4+t^6+\dots) + \sum_{k=1}^{\infty} t^k \left(\sum_{j=0}^{n-1} 2(d-1)^{k/2}T_k\left(\frac{\mu_j}{2\sqrt{d-1}}\right)\right) \\ &= n(d-2)(t^2+t^4+t^6+\dots) + \sum_{j=0}^{n-1}\left(\frac{2-\mu_j t}{1-\mu_j t+(d-1)t^2} - 2\right) \\ &= n(d-2)\frac{t^2}{1-t^2} + \sum_{j=0}^{n-1}\frac{\mu_j t - 2(d-1)t^2}{1-\mu_j t+(d-1)t^2} \end{aligned}$$

Thus,

$$N_1 + N_2 t + N_3 t^2 + \dots = n(d-2)\frac{t}{1-t^2} + \sum_{j=0}^{n-1}\frac{\mu_j - 2(d-1)t}{1-\mu_j t+(d-1)t^2}$$

While this expression does not seem very elegant stated this way, observe that the derivative of $1-t^2$ is $-2t$, and the derivative of $1-\mu_j t+(d-1)t^2$ is $-\mu_j + 2(d-1)t$. Rewriting the above expression to highlight this observation,

$$N_1 + N_2 t + N_3 t^2 + \dots = -\frac{n(d-2)}{2}\frac{-2t}{1-t^2} - \sum_{j=0}^{n-1}\frac{-\mu_j + 2(d-1)t}{1-\mu_j t+(d-1)t^2}$$

This suggests that we could integrate both sides to obtain

$$\begin{aligned} N_1 t + N_2\frac{t^2}{2} + N_3\frac{t^3}{3} + \dots &= -\frac{n(d-2)}{2}\log(1-t^2) - \sum_{j=0}^{n-1}\log(1-\mu_j t+(d-1)t^2) \\ &= -\left(\frac{nd}{2}-n\right)\log(1-t^2) - \log\left(\prod_{j=0}^{n-1} 1-\mu_j t+(d-1)t^2\right) \\ &= -(|E|-|V|)\log(1-t^2) - \log\left(det(I-At+(d-1)t^2)\right) \end{aligned}$$

Now since we know that the Ihara zeta function $\zeta_G(t)$ has the expression

$$\zeta_G = exp\left(N_1 t + N_2\frac{t^2}{2} + N_3\frac{t^3}{3} + \dots\right)$$

we now have the familiar determinant formula for the zeta function in terms of the adjacency matrix:

▶ **Theorem 9** (Bass's determinant formula). *Let $d \geq 3$ and $G = (V, E)$ be a d-regular connected graph with adjacency matrix A. Then*

$$\zeta_G(t) = \frac{(1-t^2)^{|V|-|E|}}{det(I-At+(d-1)t^2)}$$

Acknowledgements. We would like to thank Amnon Ta-Shma, Doron Zeilberger, and Dean Doron for helpful discussions.

References

1 Noga Alon, Itai Benjamini, Eyal Lubetzky, and Sasha Sodin. Non-backtracking random walks mix faster. *Communications in Contemporary Mathematics*, 9(04):585–603, 2007.

2 Hyman Bass. The ihara-selberg zeta function of a tree lattice. *International Journal of Mathematics*, 3(06):717–797, 1992.

3 Giuliana Davidoff, Peter Sarnak, and Alain Valette. *Elementary number theory, group theory and Ramanujan graphs*, volume 55. Cambridge University Press, 2003.

4 Dominique Foata and Doron Zeilberger. A combinatorial proof of bass's evaluations of the ihara-selberg zeta function for graphs. *Transactions of the American Mathematical Society*, 351(6):2257–2274, 1999.

5 Ki-ichiro Hashimoto. Zeta functions of finite graphs and representations of p-adic groups. *Automorphic forms and geometry of arithmetic varieties.*, pages 211–280, 1989.

6 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

7 Yasutaka Ihara. On discrete subgroups of the two by two projective linear group over p-adic fields. *Journal of the Mathematical Society of Japan*, 18(3):219–235, 1966.

8 Mark Kempton. Non-backtracking random walks and a weighted ihara's theorem. *arXiv preprint arXiv:1603.05553*, 2016.

9 Motoko Kotani and Toshikazu Sunada. Zeta functions of finite graphs. *J. Math. Sci. Univ. Tokyo*, 7:7–25, 2000.

10 Eyal Lubetzky and Yuval Peres. Cutoff on all ramanujan graphs. *Geometric and Functional Analysis*, 26(4):1190–1216, 2016.

11 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

12 Avinoam Mann. *How groups grow*, volume 395. Cambridge university press, 2011.

13 Adam Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families i: Bipartite ramanujan graphs of all degrees. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 529–537. IEEE, 2013.

14 Moshe Morgenstern. Existence and explicit constructions of q+ 1 regular ramanujan graphs for every prime power q. *Journal of Combinatorial Theory, Series B*, 62(1):44–62, 1994.

15 M Ram Murty. Ramanujan graphs. *Journal-Ramanujan Mathematical Society*, 18(1):33–52, 2003.

16 Alon Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991.

17 Harold M Stark and Audrey A Terras. Zeta functions of finite graphs and coverings. *Advances in Mathematics*, 121(1):124–165, 1996.

VLDL Satisfiability and Model Checking via Tree Automata*†

Alexander Weinert

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
weinert@react.uni-saarland.de

Abstract

We present novel algorithms solving the satisfiability problem and the model checking problem for Visibly Linear Dynamic Logic (VLDL) in asymptotically optimal time via a reduction to the emptiness problem for tree automata with Büchi acceptance. Since VLDL allows for the specification of important properties of recursive systems, this reduction enables the efficient analysis of such systems.

Furthermore, as the problem of tree automata emptiness is well-studied, this reduction enables leveraging the mature algorithms and tools for that problem in order to solve the satisfiability problem and the model checking problem for VLDL.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Visibly Linear Dynamic Logic, Visibly Pushdown Languages, Satisfiability, Model Checking, Tree Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.47

1 Introduction

Visibly Linear Dynamic Logic (VLDL) [24] is an expressive formalism for specifying properties of recursive systems that allows for an intuitive and modular specification of an important subclass of context-free properties. Although there exist tight bounds on the asymptotic complexity of the satisfiability and the model checking problem for VLDL properties [24], the upper bounds for both problems are witnessed by algorithms that rely on an intricate reduction of the problems to the emptiness problem for visibly pushdown automata [2], for which tool support is lacking.

We present novel reductions of the problems of VLDL satisfiability and VLDL model checking to the emptiness problem for tree automata [20], yielding asymptotically optimal algorithms for both problems. Moreover, as the emptiness problem for tree automata reduces to the problem of solving two-player games with perfect information [15], which is of great importance in the fields of program verification and program synthesis and enjoys mature tool support, the algorithms yielded by our reductions allow us to leverage this tool support for solving the problems of VLDL satisfiability and VLDL model checking.

VLDL is an extension of Linear Temporal Logic (LTL) [16], the de-facto standard for the specification of properties of non-recursive systems. Although popular, it is lacking in expressivity, as it cannot even express all ω-regular properties. The logic VLDL addresses this shortcoming by guarding the temporal operators of LTL with visibly pushdown automata (VPAs) [2]. A VPA is a pushdown automaton that operates over a fixed partition of the

* Supported by the project "TriCS" (ZI 1516/1-1) of the German Research Foundation (DFG).
† A full version of the paper is available at [23], https://arxiv.org/abs/1708.00699.

input alphabet into calls, returns, and local actions, and has to push (pop) a symbol onto (off) its stack whenever it reads a call (return). Upon processing local actions, the automaton must not touch the stack. Due to this extension, VLDL characterizes exactly the visibly pushdown languages over infinite words [24].

A VLDL formula can be compiled into an equivalent VPA over infinite words of exponential size [24]: First, the VLDL formula is translated into a one-way alternating jumping automaton (1-AJA) [5], an automaton without stack that is able to jump from a call to its matching return. This automaton is then transformed into a VPA of exponential size [5]. Since each visibly pushdown automaton is a classical pushdown automaton, the emptiness problem for VPAs is decidable in polynomial time [2]. The translation from 1-AJAs to VPAs, however, is quite involved, as it handles a far more complex model than is needed for the translation of VLDL formulas into 1-AJAs, thus hampering efforts towards an implementation of the translation from VLDL to VPAs. This effort is further encumbered by the scant availability of emptiness checkers and of model checkers for pushdown systems.

In this work, we introduce novel algorithms solving both the emptiness problem and the model checking problem for VLDL formulas in asymptotically optimal time using a translation of VLDL formulas to nondeterministic tree automata with Büchi acceptance. The technical core of this translation is formed by an encoding of words over visibly pushdown alphabets into trees that is adapted from the encoding of such words given by Alur and Madhusudan [2], as well as by a translation of the 1-AJAs constructed from VLDL formulas into tree automata using an adaptation of the breakpoint-construction by Miyano and Hayashi [14] in order to remove alternation and obtain a nondeterministic automaton. We then check satisfiability of a VLDL formula by checking the resulting tree automaton for emptiness. For model checking a visibly pushdown system against a VLDL specification, we translate the negation of the specification as well as the visibly pushdown system into tree automata, which we subsequently intersect and check for emptiness.

Thus, we reduce both the satisfiability and the model checking problem for VLDL to the emptiness problem for nondeterministic tree automata with Büchi acceptance. Hence, we reduce the complex formalism of VLDL to the simple model of nondeterministic tree automata. Moreover, since the problem of tree automata emptiness reduces to that of solving Büchi games, which is efficiently solvable [7, 17] and enjoys mature tool support [9, 10], our novel reductions enable efficient implementations of satisfiability checkers and of model checkers for VLDL.

Related Work. There exist a number of logics other than VLDL that characterize the class of visibly pushdown languages over infinite words, most prominently Visibly Linear Temporal Logic (VLTL) [6], a fixed-point logic [5], and monadic second order logic augmented with a binary matching predicate (MSO_μ) [2]. We focus here on the logic VLDL, as it most naturally extends the concepts used by LTL [16], the de-facto standard for the specification of non-recursive properties.

Moreover, there exist tools for model checking recursive systems, e.g., Bebop [3, 4] and Moped [18, 19]. These tools are, however, no longer under active development, and have, to the best of our knowledge, not found widespread adoption. In combination with the intricate translation of 1-AJAs into VPAs, this motivates the development of the novel translation of VLDL formulas into tree automata presented in this work.

A number of problems have been reduced to the emptiness problem for tree automata, as such automata are a natural model for capturing the branching-time behavior of systems [15]. Moreover, the theory of tree automata is well-studied, with the most famous result being

equivalence of tree automata and monadic second order logic of two successors [21, 25]. Finally, the emptiness problem for tree automata with Büchi acceptance reduces to the problem of solving two-player Büchi games with perfect information [8]. Such games can be solved efficiently [17] and, since Büchi games are a special case of parity games, there exist mature solvers for them [9, 10].

Our Contributions. Firstly, in Section 3 we adapt the tree-encoding of words over visibly pushdown alphabets introduced by Alur and Madhusudan [2] to our setting and show that the resulting trees are recognizable by a tree automaton with Büchi acceptance condition in Theorem 1.

Secondly, in Section 4, we show how to construct tree automata recognizing the encodings of all words satisfying a given VLDL formula in Theorem 2. Moreover, we show that the resulting automaton is of exponential size[1] measured in the size of the original formula and that this translation yields an asymptotically optimal algorithm for satisfiability checking of VLDL formulas.

Finally, in Section 5 we provide a translation of visibly pushdown systems into tree automata recognizing the encodings of all traces of the system. When combining this with the results from Section 4, we obtain an asymptotically optimal algorithm for model checking visibly pushdown systems against VLDL specifications. This result is given in Theorem 6.

2 Preliminaries

In this section we introduce the basic notions used in the remainder of this work, namely (nondeterministic) visibly pushdown automata and related concepts [2].

2.1 Visibly Pushdown Languages

A pushdown alphabet $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$ is a partition of an alphabet Σ into calls Σ_c, returns Σ_r, and local actions Σ_l. We write w and α for finite and infinite words, respectively, and inductively define the stack height $sh(w)$ reached by any automaton after reading w as $sh(\varepsilon) = 0$, $sh(wc) = sh(w) + 1$ for $c \in \Sigma_c$, $sh(wr) = \max\{0, sh(w) - 1\}$ for $r \in \Sigma_r$, and $sh(wl) = sh(w)$ for $l \in \Sigma_l$. Let $\alpha = \alpha_0\alpha_1\alpha_2\cdots$ be a finite or infinite word. We say that a call $\alpha_i \in \Sigma_c$ at some position i of α is matched if there exists a position $j > i$ such that $\alpha_j \in \Sigma_r$ and $sh(\alpha_0 \cdots \alpha_{i-1}) = sh(\alpha_0 \cdots \alpha_j)$ and call the return at the smallest such position j the matching return of c. If no such j exists, we call c an unmatched call. If α_i is a matched call with α_j as its matching return, we call the infix $\alpha_{i+1} \cdots \alpha_{j-1}$ of α the nested infix of position i. A word is well-matched if it does not contain a return that is not a matching return. Well-matched words may, however, contain unmatched calls.

A visibly pushdown system (VPS) $\mathcal{S} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, q_I)$ consists of a finite set Q of states, a pushdown alphabet $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$, a stack alphabet Γ, which contains a stack-bottom marker $\bot$, a transition relation $\Delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\bot\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_l \times Q)$, and an initial state $q_I \in Q$. A configuration (q, γ) of $\mathcal{S}$ is a pair of a state $q \in Q$ and a stack content $\gamma \in \Gamma_c = (\Gamma \setminus \{\bot\})^* \cdot \bot$. The VPS $\mathcal{S}$ induces the configuration graph $G_{\mathcal{S}} = (Q \times \Gamma_c, E)$ with $E \subseteq ((Q \times \Gamma_c) \times \Sigma \times (Q \times \Gamma_c))$ and $((q, \gamma), a, (q', \gamma')) \in E$ if and only if either (i) $a \in \Sigma_c$, $(q, a, q', A) \in \Delta$, and $A\gamma = \gamma'$, (ii) $a \in \Sigma_r$, $(q, a, \bot, q') \in \Delta$, and $\gamma = \gamma' = \bot$, (iii)

[1] Throughout this work, we say that $f(x)$ is exponential in x if there exist a constant c and a polynomial p such that $f(x) = c^{p(x)}$.

$a \in \Sigma_r$, $(q, a, A, q') \in \Delta$, $A \neq \bot$, and $\gamma = A\gamma'$, or (*iv*) $a \in \Sigma_l$, $(q, a, q') \in \Delta$, and $\gamma = \gamma'$. A run $\pi = (q_0, \gamma_0) \cdots (q_n, \gamma_n)$ of $\mathcal{S}$ on $w = w_0 \cdots w_{n-1} \in \Sigma^*$ is a sequence of configurations where $q_0 = q_I$, $\gamma_0 = \bot$, and where $((q_i, \gamma_i), w_i, (q_{i+1}, \gamma_{i+1})) \in E$ in $G_{\mathcal{S}}$ for all $i \in [0; n-1]$. Infinite runs of $\mathcal{S}$ on infinite words are defined analogously. We define $\mathit{traces}(\mathcal{S})$ as the set of all infinite words α for which there exists a run of $\mathcal{S}$ on α.

2.2 Visibly Linear Dynamic Logic

Let P be a finite set of atomic propositions and let $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$ be a partition of $\Sigma = 2^P$. The syntax of Visibly Linear Dynamic Logic (VLDL) [24] is defined by the grammar

$$\varphi = p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\mathfrak{A}\rangle\varphi \mid [\mathfrak{A}]\varphi \ ,$$

where $p \in P$ and $\mathfrak{A}$ ranges over testing visibly pushdown automata (TVPA) over the fixed alphabet $\widetilde{\Sigma}$. A TVPA $\mathfrak{A} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, q_I, Q_F, t)$ consists of a VPS $\mathcal{S} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, q_I)$, a set of final states $Q_F \subseteq Q$, and a function t mapping states to VLDL formulas over $\widetilde{\Sigma}$. We define $|\mathfrak{A}| = |Q| + |\Gamma|$ and $|\varphi|$ as the sum of $|\mathrm{cl}(\varphi)|$ and the sum of the sizes of the automata contained in φ, where $\mathrm{cl}(\varphi)$ is the set of all subformulas of φ, including those contained as tests in automata and their subformulas. We require this subformula-relation to be non-circular. A run of $\mathfrak{A}$ on a finite word w is a run of the underlying VPS $\mathcal{S}$ on w. Such a run is accepting if its final state is in Q_F.

Let φ be a VLDL formula, let $\alpha = \alpha_0\alpha_1\alpha_2 \cdots \in \Sigma^\omega$ and let $i \in \mathbb{N}$ be a position in α. We define the semantics of φ in the straightforward way for atomic propositions and for Boolean connectives. Furthermore, we define

- $(\alpha, i) \models \langle\mathfrak{A}\rangle\varphi$ if there exists $j \geq i$ s.t. $(i, j) \in \mathcal{R}_{\mathfrak{A}}(\alpha)$ and $(\alpha, j) \models \varphi$,
- $(\alpha, i) \models [\mathfrak{A}]\varphi$ if for all $j \geq i$, $(i, j) \in \mathcal{R}_{\mathfrak{A}}(\alpha)$ implies $(\alpha, j) \models \varphi$,

with $\mathcal{R}_{\mathfrak{A}}(\alpha) = \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid \exists \text{ acc. run } (q_0, \sigma_0) \cdots (q_{j-i}, \sigma_{j-i}) \text{ of } \mathfrak{A} \text{ on } \alpha_i \cdots \alpha_{j-1} \text{ and } \forall k \in [0; j-i].\ (\alpha, i+k) \models t(q_k)\}$. We write $\alpha \models \varphi$ as a shorthand for $(\alpha, 0) \models \varphi$ and say that α is a model of φ in this case. The language of φ is defined as $L(\varphi) = \{\alpha \in \Sigma^\omega \mid \alpha \models \varphi\}$. If $\mathcal{L}(\varphi) \neq \emptyset$, we say that φ is satisfiable.

2.3 Tree Automata

Let $\mathbb{B} = \{0, 1\}$ and let Σ be an alphabet. A Σ-tree t is a mapping $t\colon \mathbb{B}^* \to \Sigma$. We call a finite word $b \in \mathbb{B}^*$ a node and an infinite word $\beta \in \mathbb{B}^\omega$ a branch. Given a node b, we call the nodes $b0$ and $b1$ the left- and right-hand children of b. Analogously, we call the trees rooted at the left- and right-hand children of b the left- and right-hand subtrees of b, respectively. Moreover, b is the parent of both $b0$ and $b1$. The node ε is the root of t. As each node b is associated with the unique path from the root of the tree to b, we say that a node b' is on the path to b if b' is a prefix of b. Similarly, we say that a branch β contains a node b if b is a prefix of β. If $t(b) = a$, we say that b is labeled with a. Moreover, given a tree t and a node b, we define the sub-tree $t\big|_b$ of t rooted at b by $t\big|_b(b') = t(bb')$.

A tree automaton (with Büchi acceptance) $\mathfrak{T} = (Q, \Sigma, \Delta, q_I, Q_F)$ consists of a finite set of states Q, an alphabet Σ, a transition relation $\Delta \subseteq Q \times \Sigma \times Q \times Q$, an initial state $q_I \in Q$, and a set of accepting states $Q_F \subseteq Q$. A run r of $\mathfrak{T}$ on a Σ-tree t is a Q-tree with $r(\varepsilon) = q_I$ and $(r(b), t(b), r(b0), r(b1)) \in \Delta$ for all $b \in \mathbb{B}^*$. A branch of r is accepting if it contains infinitely many nodes b with $r(b) \in Q_F$. A run is accepting if all of its branches are accepting, while an automaton $\mathfrak{T}$ accepts a tree t if there exists an accepting run of $\mathfrak{T}$ on t. The language $L(\mathfrak{T})$ of $\mathfrak{T}$ is defined as the set of all trees accepted by $\mathfrak{T}$. A set of trees is regular if there exists a tree automaton recognizing it. We define $|\mathfrak{T}| = |Q|$. Tree automata are

$$\mathrm{st}(cwr\alpha) = \begin{array}{ccc} & c & \\ \swarrow & & \searrow \\ \mathrm{st}(r\alpha) & & \mathrm{st}(w) \end{array} \quad \text{if } c \in \Sigma_c \text{ and } r \text{ is matching return of } c$$

$$\mathrm{st}(c\alpha) = \begin{array}{ccc} & c & \\ \swarrow & & \searrow \\ \mathrm{st}(\varepsilon) & & \mathrm{st}(\alpha) \end{array} \quad \text{if } c \in \Sigma_c \text{ and } c \text{ is unmatched}$$

$$\mathrm{st}(x\alpha) = \begin{array}{ccc} & x & \\ \swarrow & & \searrow \\ \mathrm{st}(\alpha) & & \mathrm{st}(\varepsilon) \end{array} \quad \text{if } x \in \Sigma_l \cup \Sigma_r$$

$$\mathrm{st}(\varepsilon) = \begin{array}{ccc} & \bot & \\ \swarrow & & \searrow \\ \mathrm{st}(\varepsilon) & & \mathrm{st}(\varepsilon) \end{array}$$

Figure 1 Definition of $\mathrm{st}\colon \Sigma^\omega \cup \Sigma^* \to T_{\Sigma_\bot}$.

closed under intersection via an adaptation of the product-construction for the intersection of automata on words. Hence, for tree automata $\mathfrak{T}_1$,$\mathfrak{T}_2$ there exists a tree automaton $\mathfrak{T}$ with $|\mathfrak{T}| \in \mathcal{O}(|\mathfrak{T}_1||\mathfrak{T}_2|)$ such that $\mathcal{L}(\mathfrak{T}) = \mathcal{L}(\mathfrak{T}_1) \cap \mathcal{L}(\mathfrak{T}_2)$.

3 Stack Trees

Alur and Madhusudan showed how to encode words over some visibly pushdown alphabet as trees by "folding away" the nested infixes of calls into subtrees, thus moving a matched call and its matching return next to each other in the resulting tree [2]. In this section, we slightly adapt their encoding in order to simplify our construction of tree automata later on in Section 4. In that section, we construct for each VLDL formula φ a tree automaton that accepts precisely the encodings of words satisfying φ.

For the remainder of this work, we fix some finite set P of atomic propositions and a pushdown alphabet $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$ as a partition of 2^P. Let $\alpha \in \Sigma^\omega$ be an infinite word and define $\Sigma_\bot = \Sigma \cup \{\bot\}$, where $\bot$ is some fresh symbol. Intuitively, every node in the resulting $\Sigma_\bot$-tree either denotes one position of α, or it is labeled with the special symbol $\bot$. Formally, we define the function st mapping finite and infinite words over Σ to infinite $\Sigma_\bot$-trees in Figure 1. At every matched call, we encode its matched infix and the suffix starting at and including its matched return in the right- and left-hand subtrees, respectively. At an unmatched call, we encode the infinite suffix of the word starting at the symbol succeeding the call in the right-hand subtree. If the current letter is not a call, we encode the suffix starting at the current letter's successor in the left-hand subtree. All vertices not encoding a symbol of α are labeled with $\bot$.

A $\Sigma_\bot$-tree t is a stack tree if $t = \mathrm{st}(\alpha)$ for some $\alpha \in \Sigma^\omega$. We define the set of all stack trees over Σ as $\mathrm{st}(\Sigma^\omega) = \{\mathrm{st}(\alpha) \mid \alpha \in \Sigma^\omega\}$.

▶ **Theorem 1.** *The set* $\mathrm{st}(\Sigma^\omega)$ *is regular.*

Proof. We first introduce some notation. Let t be a $\Sigma_\bot$-tree. We say that a node b is a matched call if $t(b) \in \Sigma_c$ and $t(b0) \in \Sigma_r$. Similarly, the node $b0$ is a matched return if $t(b) \in \Sigma_c$. If all calls and returns in t are matched, we say that t is well-matched. In contrast to the notion of well-matched words, we demand that a well-matched tree does not contain unmatched calls. Furthermore, we call a branch β finite in t if it eventually only contains $\bot$-labeled vertices. Otherwise, we call β infinite in t. Moreover, we call a tree finite if all of its branches are finite.

We claim that a $\Sigma_\bot$-tree t is a stack tree if and only if $t(\varepsilon) \neq \bot$, if there exists a single branch that is infinite in t, and if the following properties hold true for all $b \in \mathbb{B}^*$:

1. If $t(b) = \bot$, then $t(b0) = t(b1) = \bot$,

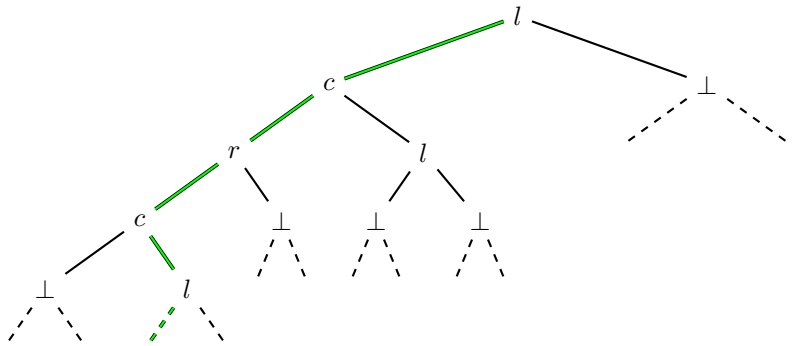


Figure 2 Encoding of $\alpha = lclrcl\cdots$, where the second occurrence of c is unmatched. The cardinal branch of $\mathrm{st}(\alpha)$ is marked in green and doubly lined.

2. if $t(b) \in \Sigma_c$, then
   a. if b is matched, then $t\big|_{b1}$ is finite and well-matched, and
   b. if b is unmatched, then $t(b0) = \bot$ and $t\big|_{b1}$ contains no unmatched returns, and
3. if $t(b) \in \Sigma_l \cup \Sigma_r$, then $t(b1) = \bot$.

Each of these properties can be checked by a tree automaton. As tree automata are closed under intersection, we can construct a single tree automaton that checks all of the above properties. In the full version, we show that the conditions above indeed characterize $\mathrm{st}(\Sigma^\omega)$ [23]. ◀

From the proof of Theorem 1 we obtain that for each $\alpha \in \Sigma^\omega$, there exists a unique branch $\beta = b_0 b_1 b_2 \cdots$ of $\mathrm{st}(\alpha)$ such that $\mathrm{st}(\alpha)(b_0 \cdots b_{i-1}) \neq \bot$ for each $i \in \mathbb{N}$. We call β the cardinal branch of $\mathrm{st}(\alpha)$ and we call the positions of the symbols encoded along β the cardinal positions of α.

We give an example of the tree-encoding of the word $\alpha = lclrcl\cdots$ over the alphabet $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l) = (\{c\}, \{r\}, \{l\})$ in Figure 2. The positions $0, 1, 3, 4$ are cardinal positions of α. Moreover, if we assume the second c in α to be unmatched, then the position 5 is a cardinal position as well.

Recall that we defined $sh(w)$ to be the stack height reached by any visibly pushdown automaton after processing $w \in \Sigma^*$. Löding et al. defined the steps of a word $\alpha = \alpha_0\alpha_1\alpha_2\cdots$ as those positions of α that reach a lower bound on the stack height reached during processing the remainder of the word, i.e., $steps(\alpha) = \{i \mid \forall j \geq i.\, sh(\alpha_0 \cdots \alpha_i) \leq sh(\alpha_0 \cdots \alpha_j)\}$ [13]. This allows for an alternative characterization of cardinal positions: A position i is a cardinal position of α if and only if it is either a step, or if α_i is the matching return of some call occurring at a step.

4 Reducing VLDL Satisfiability to Tree Automata Emptiness

We now reduce the problem of VLDL satisfiability to the emptiness problem for tree automata. The former problem is formulated as follows: "Given some VLDL formula φ, is φ satisfiable?" We formalize the reduction of this problem to the emptiness problem for tree automata in the following theorem:

▶ **Theorem 2.** *For every VLDL formula φ there exists an effectively constructible tree automaton $\mathfrak{T}$ such that $\mathcal{L}(\mathfrak{T}) = \mathrm{st}(\mathcal{L}(\varphi))$ with $|\mathfrak{T}| \in \mathcal{O}(2^{4|\varphi|^3})$.*

Due to this theorem, we obtain an algorithm that checks VLDL formulas for satisfiability by first transforming a given formula φ into the tree automaton $\mathfrak{T}$ recognizing $\mathrm{st}(\mathcal{L}(\varphi))$ and subsequently checking $\mathcal{L}(\mathfrak{T})$ for emptiness. Since tree automata can be checked for emptiness in polynomial time [11, 17], the algorithm runs in exponential time in $|\varphi|$. As the problem of deciding VLDL satisfiability is ExpTime-hard [24], this algorithm is asymptotically optimal.

We split the proof of Theorem 2 into two parts: First, we transform a given VLDL formula into an equivalent one-way alternating jumping automaton (1-AJA) [5] of polynomial size. A 1-AJA is an alternating finite-state automaton on words that is able to "jump" from calls to their matching return, skipping the nested infix. We describe this construction in the proof of Lemma 3. In a second step, we transform the obtained 1-AJA $\mathfrak{A}$ into a tree automaton of exponential size that recognizes the stack trees of words recognized by $\mathfrak{A}$. We describe this construction in the proof of Lemma 5.

Let us first define the above mentioned 1-AJA [5] formally. First, let $\mathit{Dirs} = \{\rightarrow, \curvearrowright\}$, where we use $\rightsquigarrow$ to denote an arbitrary member of Dirs. Moreover, for a finite set Q and $\rightsquigarrow \in \mathit{Dirs}$, let $\mathit{Comms}(Q) = \mathit{Dirs} \times Q \times Q$ and let $\mathcal{B}^+(\mathit{Comms}(Q))$ be the set of positive Boolean formulas over $\mathit{Comms}(Q)$. Note that $\mathcal{B}^+(\mathit{Comms}(Q))$ does not include the shorthands *true* nor *false*. A 1-AJA (with Büchi acceptance) $\mathfrak{A} = (Q, \widetilde{\Sigma}, \delta, q_I, Q_F)$ consists of a finite set of states Q, a visibly pushdown alphabet $\widetilde{\Sigma}$, a transition function $\delta\colon Q \times \Sigma \to \mathcal{B}^+(\mathit{Comms}(Q))$, an initial state $q_I \in Q$, and a set of accepting states $Q_F \subseteq Q$. We define $|\mathfrak{A}| = |Q|$.

Intuitively, when such an automaton $\mathfrak{A}$ is in state q at position i of the word $\alpha = \alpha_0\alpha_1\alpha_2\cdots$, it guesses a set of commands $C \subseteq \mathit{Comms}(Q)$ such that $C \models \delta(q, \alpha_i)$. It then spawns one copy of itself for each command $(\rightsquigarrow, q_\rightarrow, q_\curvearrowright) \in C$ and executes the command with that copy. If $\rightsquigarrow = \curvearrowright$ and if α_i is a matched call, the copy jumps to the position of the matching return of α_i and transitions to state $q_\curvearrowright$. Otherwise, i.e., if $\rightsquigarrow = \rightarrow$, or if α_i is not a matched call, the automaton advances to position $i+1$ and transitions to state $q_\rightarrow$. All copies of $\mathfrak{A}$ proceed in parallel. The automaton $\mathfrak{A}$ accepts a word if all of its copies it visit accepting states infinitely often.

Formally, a run of $\mathfrak{A}$ on an infinite word $\alpha = \alpha_0\alpha_1\alpha_2\cdots$ is an infinite directed acyclic graph $R = (V, E)$ with $V \subseteq \mathbb{N} \times Q$, where $v_I = (0, q_I) \in V$ and all $v \in V$ are reachable from v_I. We call v_I the initial vertex of R and say that a vertex $(i, q) \in V$ is on level i of R. We require that for each $(i, q) \in V$, there exists some $C \subseteq \mathit{Comms}(Q)$ such that $C \models \delta(q, \alpha_i)$ and such that $((i, q), (j, q')) \in E$ if and only if $(j, q') = \mathit{app}(i, c)$ for some $c \in C$. To this end, the command-application function app is defined as $\mathit{app}(i, (\rightsquigarrow, q_\rightarrow, q_\curvearrowright)) = (j, q_\curvearrowright)$ if $\rightsquigarrow = \curvearrowright$ and α_i is a matched call with α_j as its matching return, and $\mathit{app}(i, (\rightsquigarrow, q_\rightarrow, q_\curvearrowright)) = (i+1, q_\rightarrow)$ otherwise. We say that a vertex (i, q) is accepting if q is accepting. Furthermore, a run R is accepting if each vertex in R has at least one successor and if all infinite paths through R starting in v_I contain infinitely many accepting vertices.

In contrast to the classical definition of runs of alternating automata (without jumping capability), an edge in a run of a 1-AJA does not characterize an advance by a single symbol. Instead, there exist "long" edges that characterize the automaton skipping a nested infix. Thus, there may exist positions i such that a run of a 1-AJA on a word does not contain any vertices of the form (i, q), since all copies of the automaton jump over position i. The cardinal positions of a word α, however, serve as synchronization points of a run on α, as no copy of the automaton is able to jump over such positions.

▶ **Lemma 3.** *For every VLDL formula φ there exists an effectively constructible 1-AJA $\mathfrak{A}$ such that $\mathcal{L}(\mathfrak{A}) = \mathcal{L}(\varphi)$ and such that $|\mathfrak{A}| \in \mathcal{O}(|\varphi|^3)$.*

Proof. In earlier work, we constructed a 1-AJA with parity acceptance condition from a given VLDL formula φ by induction over the structure of φ [24]. This more complicated

acceptance condition allowed for complementation without a state-space-blowup in the construction of an automaton equivalent to $\varphi = \neg\varphi'$. As we are now aiming for a 1-AJA with a simpler acceptance condition, namely a Büchi condition, we adapt this previous construction.

In order to prevent the costly complementation of 1-AJA (with Büchi acceptance), we require φ to be in negation normal form (NNF), i.e., we assume that negations only occur directly preceding atomic propositions. Should this not be the case, we can easily transform φ into NNF by "pushing down" negations along the syntax tree, using De Morgan's law and the duality $\neg\langle\mathfrak{A}\rangle\varphi \equiv [\mathfrak{A}]\neg\varphi$. Note that this latter duality does not require complementation of the automaton $\mathfrak{A}$, hence it is applicable in constant time. We then construct $\mathfrak{A}_\varphi$ equivalent to φ inductively over the structure of φ.

If $\varphi = p$, $\varphi = \neg p$, or $\varphi = \varphi_1 \circ \varphi_2$ for $\circ \in \{\vee, \wedge\}$, we trivially obtain $\mathfrak{A}_\varphi$, with $|\mathfrak{A}_p| = |\mathfrak{A}_{\neg p}| \in \mathcal{O}(1)$ and $|\mathfrak{A}_{\varphi_1 \circ \varphi_2}| \in \mathcal{O}(|\mathfrak{A}_{\varphi_1}| + |\mathfrak{A}_{\varphi_2}|)$ due to closure of 1-AJA under these operations [5]. If $\varphi = \langle\mathfrak{A}\rangle\varphi'$, we follow the same intuition as in the previous construction, i.e., we construct the 1-AJA $\mathfrak{A}_\varphi$ such that a single copy of $\mathfrak{A}$ jumps along the cardinal positions of the input-word and spawns copies at every matched call in order to verify that the jumps taken correctly summarize finite runs of $\mathfrak{A}$ on the nested infix. Additionally, $\mathfrak{A}_\varphi$ spawns copies verifying that the tests annotating the states along the simulated run hold true. Finally, $\mathfrak{A}_\varphi$ nondeterministically decides to transition into $\mathfrak{A}_{\varphi'}$. The complete construction for this case can be found in the full version of our previous work [24]. Although we constructed 1-AJAs with parity acceptance condition in that work, this previous construction can easily be adapted to use Büchi acceptance by making none of the states simulating $\mathfrak{A}$ accepting in $\mathfrak{A}_\varphi$, thus forcing the simulated run to eventually transition into $\mathfrak{A}_{\varphi'}$.

If $\varphi = [\mathfrak{A}]\varphi'$, we obtain an automaton equivalent to φ via a dual construction to the previous case $\varphi = \langle\mathfrak{A}\rangle\varphi'$. We provide the detailed construction for this case in the full version of this work [23]. ◀

Having translated VLDL formulas into 1-AJAs, we now show how to transform a given 1-AJA $\mathfrak{A}$ into a tree automaton recognizing the stack trees of words from $\mathcal{L}(\mathfrak{A})$. To this end, consider a run R of a 1-AJA $\mathfrak{A}$ on some word $\alpha \in \Sigma^\omega$, as illustrated on the left-hand side of Figure 3. As stated above, the cardinal positions of the processed word serve as synchronization points in the run of $\mathfrak{A}$ on α: If i is a cardinal position of α, then there exist no positions $j, j' \in \mathbb{N}$ with $j < i < j'$ such that R contains an edge from level j to level j'. In other words, each infinite path starting in the initial vertex v_I of R contains a vertex on level i for each cardinal position i of α. Hence, we are able to decide whether or not R is accepting by considering finite paths of R starting and ending in levels i and i', respectively, where i and i' are adjacent cardinal positions of α.

More formally, we demonstrate that the breakpoint construction of Miyano and Hayashi [14] can be adapted to 1-AJAs. To this end, let $R = (V, E)$ be a run of some 1-AJA on some word. A breakpoint sequence over R is an infinite sequence of cardinal positions $0 = i_0 < i_1 < i_2 \cdots$ of α such that all finite paths in R starting on level i_j and ending on level i_{j+1} contain at least one accepting vertex. Each i_j in a breakpoint sequence is called a breakpoint.

▶ **Lemma 4.** *Let R be a run of a 1-AJA. The run R is accepting if and only if there exists a breakpoint sequence over R.*

Proof. First assume that there exists a breakpoint sequence $0 = i_0, i_1, i_2, \ldots$ over R. Then R is clearly accepting, as each infinite path π of R starting in v_I is of the form $\pi =$

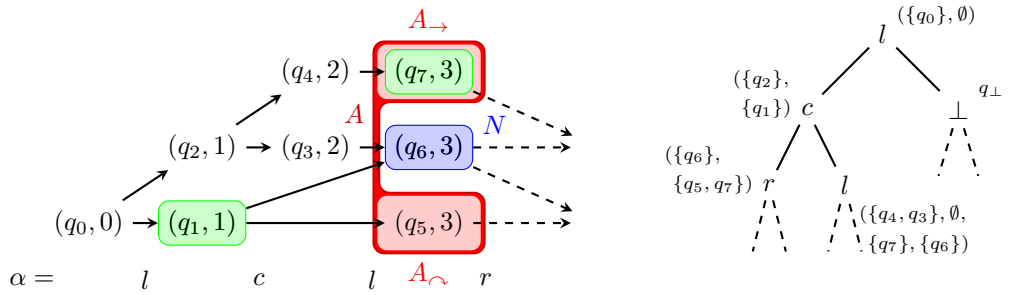


Figure 3 Encoding of a run of a 1-AJA (left) into a run of a tree automaton (right). The states q_1 and q_7 are accepting and marked in green. The positions 0, 1, and 3 are cardinal positions of α. We have $A_\rightarrow = \{q_5\}$, $A_\frown = \{q_7\}$, $A_3 = \{q_5, q_7\}$, and $N_\rightarrow = N_\frown = N = \{q_6\}$. The set A is marked in dark red, while the sets $A_\rightarrow$ and $A_\frown$ are marked in light red. The set N is marked in blue.

$v_0\pi_0v_1\pi_1v_2\pi_2\cdots$, where each $v_j\pi_jv_{j+1}$ is a path from level i_j to level i_{j+1}, hence $v_j\pi_jv_{j+1}$ contains at least one accepting vertex. Thus, π is accepting.

For the other direction, assume that R is accepting. We show the existence of a breakpoint sequence inductively and begin by defining $i_0 = 0$. Now let $i_0, \ldots, i_j$ be a finite prefix of a breakpoint sequence and assume towards a contradiction that no cardinal position i_{j+1} exists such that $i_0, \ldots, i_j, i_{j+1}$ is a prefix of a breakpoint sequence.

Consider the subgraph of R induced by removing those accepting vertices occurring on levels greater than i_j from R. Since for each cardinal position k, there exists a path from some vertex on level i_j to some vertex on level k that does not contain an accepting vertex, this graph is infinite. Moreover, it is finitely branching, since it is a subgraph of the finitely branching graph R. Due to König's Lemma, the induced subgraph contains an infinite path.

Hence, there also exists an infinite path starting on level i_j that does not contain an accepting vertex, which contradicts R being accepting. Thus, there exists a cardinal position i_{j+1} such that $i_0, \ldots, i_j, i_{j+1}$ is a prefix of some breakpoint sequence. Hence, there exists a breakpoint sequence over R. ◀

Given some 1-AJA $\mathfrak{A}$, we now construct a tree automaton that verifies that the input tree is indeed a stack tree and, if this is the case, simulates a run of $\mathfrak{A}$ on the word represented by the input tree by keeping track of the set of states at each level. Moreover, it verifies the existence of a breakpoint sequence, visiting an accepting state on the cardinal branch of the processed tree every time the corresponding symbol is at a cardinal position of the input word that can continue the prefix of the breakpoint sequence constructed so far. In order to do so, we adapt the breakpoint construction by Miyano and Hayashi [14].

The key insight of this construction is that, given some breakpoint i, the vertices of any cardinal position $j > i$ can be partitioned into two sets A and N. The set A contains those states such that each finite path from some vertex on level i to some vertex on level j visits at least one accepting state, while N contains the remaining vertices on level j. We illustrate this partitioning on the left-hand side of Figure 3. If the set N is empty, then the position j continues the breakpoint sequence constructed so far. We adapt this technique in order to translate 1-AJA into tree automata by keeping track of the sets A and N along the cardinal branch of the stack tree. Upon encountering a matched call at position i, the tree automaton guesses the sets A and N reached at the next cardinal position j and verifies this guess when processing the nested infix of position i.

▶ **Lemma 5.** *For every 1-AJA $\mathfrak{A}$ there exists an effectively constructible tree automaton $\mathfrak{T}$ such that $\mathcal{L}(\mathfrak{T}) = \mathrm{st}(\mathcal{L}(\mathfrak{A}))$ with $|\mathfrak{T}| \in \mathcal{O}(2^{4|\mathfrak{A}|})$.*

Proof. We construct a tree automaton $\mathfrak{T}'$ such that $\mathcal{L}(\mathfrak{T}') \cap \mathrm{st}(\Sigma^\omega) = \mathrm{st}(\mathcal{L}(\mathfrak{A}))$. Recall that, due to Theorem 1, we obtain a tree automaton $\mathfrak{T}_\Sigma$ with $\mathcal{L}(\mathfrak{T}_\Sigma) = \mathrm{st}(\Sigma^\omega)$. By intersecting $\mathfrak{T}'$ with $\mathfrak{T}_\Sigma$ we subsequently obtain $\mathfrak{T}$ with the properties stated above.

We have explained the behavior of the automaton $\mathfrak{T}'$ along the cardinal branch above. It remains to take into account the effect of nested infixes on the states reached by $\mathfrak{A}$ at cardinal positions. To this end, we observe that each state reached at a cardinal position is either reached by taking a jumping transition from the previous cardinal position, or by taking a direct transition from the directly preceding position, i.e., from the last position of the nested infix. Thus, when reading a matched call at position i, the automaton $\mathfrak{T}$ guesses sets $A_\rightarrow, N_\rightarrow \subseteq Q$ that are reached eventually by copies of the automaton that process the nested infix w of position i. It moreover guesses sets $A_\curvearrowright, N_\curvearrowright \subseteq Q$ that are reached by copies of the automaton that jump over the nested infix of position i. The automaton then assumes that the states in $A = A_\rightarrow \cup A_\curvearrowright$ and $N = N_\rightarrow \cup N_\curvearrowright$ are indeed reached by processing w and by skipping over w and verifies the former guess while processing $\mathrm{st}(w)$, i.e., the right-hand subtree of the matched call. The latter guess is verified using the transition function of $\mathfrak{A}$. We show an example of this encoding of a run of $\mathfrak{A}$ as a run of $\mathfrak{T}'$ on the right-hand side of Figure 3.

We use two kinds of states in order to implement this idea. States of the form (A, N), where A and N partition a nonempty subset of Q are used along the cardinal branch of the processed stack tree, implementing the breakpoint construction. Furthermore, we use states of the form $((A, N), (A_G, N_G))$, where (A, N) as well as (A_G, N_G) are partitions of nonempty subsets of Q, in order to verify the guesses A_G and N_G about the effects of processing nested infixes. Moreover, we use a sink-state $q_\perp$ in order to process subtrees labeled exclusively with $\perp$.

The detailed construction of the tree automaton $\mathfrak{T}$ recognizing stack trees of words in $\mathcal{L}(\mathfrak{A})$ can be found in the full version [23]. Correctness of $\mathfrak{T}$ follows from Lemma 4 and the above arguments. Moreover, we indeed obtain $|\mathfrak{T}| \in \mathcal{O}(2^{4|\mathfrak{A}|})$. ◀

The proof of Theorem 2 follows from Lemma 3 and Lemma 5: Given a VLDL formula φ, we first construct the 1-AJA $\mathfrak{A}$ with $\mathcal{L}(\mathfrak{A}) = \mathcal{L}(\varphi)$ as demonstrated in the proof of Lemma 3. The automaton $\mathfrak{A}$ is of size $\mathcal{O}(|\varphi|^3)$. We then construct the tree automaton $\mathfrak{T}$ with $\mathcal{L}(\mathfrak{T}) = \mathrm{st}(\mathcal{L}(\mathfrak{A}))$ as shown in the proof of Lemma 5. The automaton $\mathfrak{T}$ recognizes $\mathrm{st}(\mathcal{L}(\mathfrak{A})) = \mathrm{st}(\mathcal{L}(\varphi))$ and is of size $\mathcal{O}(2^{4|\mathfrak{A}|}) = \mathcal{O}(2^{4|\varphi|^3})$.

5 Reducing VLDL Model Checking to Tree Automata Emptiness

In the previous section we have reduced the problem of VLDL satisfiability checking to the emptiness problem for tree automata. We now consider the problem of VLDL model checking, which is formulated as follows: "Given a VPS $\mathcal{S}$ and a VLDL formula φ, does $\mathit{traces}(\mathcal{S}) \subseteq \mathcal{L}(\varphi)$ hold true?" We now reduce this problem to the emptiness problem for tree automata similarly to the reduction of the satisfiability problem for VLDL to the same problem.

▶ **Theorem 6.** *Let $\mathcal{S}$ be a VPS with state space Q and stack alphabet Γ and let φ be a VLDL formula. There exists an effectively constructible tree automaton $\mathfrak{T}$ such that $\mathcal{L}(\mathfrak{T}) = \mathrm{st}(\mathit{traces}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi))$ with $|\mathfrak{T}| \in \mathcal{O}(2^{4|\varphi|^3}|Q|^2|\Gamma|))$.*

Proof. Recall that we can effectively construct a tree automaton $\mathfrak{T}_{\neg\varphi}$ such that $\mathcal{L}(\mathfrak{T}_{\neg\varphi}) = \mathrm{st}(\mathcal{L}(\neg\varphi))$ due to Theorem 2. We now construct a tree automaton $\mathfrak{T}_\mathcal{S}$ recognizing $\mathrm{st}(\mathit{traces}(\mathcal{S}))$.

By intersecting $\mathfrak{T}_{\mathcal{S}}$ and $\mathfrak{T}_{\neg\varphi}$ we subsequently obtain the tree automaton $\mathfrak{T}$ recognizing $\mathrm{st}(\mathit{traces}(\mathcal{S})) \cap \mathrm{st}(\mathcal{L}(\neg\varphi))$. Since, due to injectivity of st, $\mathrm{st}(\mathit{traces}(\mathcal{S})) \cap \mathrm{st}(L(\neg\varphi)) = \mathrm{st}(\mathit{traces}(\mathcal{S}) \cap L(\neg\varphi))$ holds true, we obtain the stated result.

It remains to construct $\mathfrak{T}_{\mathcal{S}}$. Similarly to the proof of Lemma 5, we first construct $\mathfrak{T}'_{\mathcal{S}}$ such that $\mathcal{L}(\mathfrak{T}'_{\mathcal{S}}) \cap \mathrm{st}(\Sigma^\omega) = \mathrm{st}(\mathit{traces}(\mathcal{S}))$. By intersecting $\mathfrak{T}'_{\mathcal{S}}$ with $\mathfrak{T}_\Sigma$ recognizing $\mathrm{st}(\Sigma^\omega)$ as constructed in the proof of Theorem 1 we then obtain the required $\mathfrak{T}_{\mathcal{S}}$. The idea behind the construction of $\mathfrak{T}'_{\mathcal{S}}$ is to simulate a run of $\mathcal{S}$ along the cardinal branch of the tree. This is straightforward in the case of local actions and unmatched calls or returns. Upon encountering a matched call, $\mathfrak{T}'_{\mathcal{S}}$ guesses the state reached by $\mathcal{S}$ upon encountering the matched return and verifies that guess on the stack tree of the nested infix.

Let $\mathcal{S} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, q_I)$. We define $\mathfrak{T}'_{\mathcal{S}} = (Q', \Sigma, \Delta', q_I, Q'_F)$ with $Q' = Q \cup (Q \times Q) \cup (Q \times \Gamma) \cup (Q \times \Gamma \times Q) \cup \{q_\bot\}$, $Q'_F = Q'$, and $\Delta' = \Delta_l \cup \Delta_{uc} \cup \Delta_{ur} \cup \Delta_{mc} \cup \Delta_{mr} \cup \Delta_v \cup \Delta_s$. The individual components of Δ' are defined as follows: We process local actions using transitions of the form $\Delta_l = \{(q, l, q', q_\bot), ((q, q_G), l, (q', q_G), q_\bot) \mid (q, l, q') \in \Delta, q_G \in Q\}$. Similarly, upon encountering unmatched calls or returns, we use transitions of the form $\Delta_{uc} = \{(q, c, q_\bot, q') \mid (q, c, q', A) \in \Delta\}$ and $\Delta_{ur} = \{(q, r, q', q_\bot) \mid (q, r, \bot, q') \in \Delta\}$, respectively. When encountering a matched call, we guess a state q_G reached by the automaton upon processing the matching return and verify that guess using transitions from $\Delta_{mc} = \{(q, c, (q_G, A), (q', q_G)) \mid (q, c, q', A) \in \Delta, q_G \in Q\} \cup \{((q, q_G), c, (q'_G, A, q_G), (q', q'_G)) \mid (q, c, q', A) \in \Delta, q_G, q'_G \in Q\}$. Upon encountering a matched return, we are in some state from $(Q \times \Gamma) \cup (Q \times \Gamma \times Q)$, since a matched return only occurs directly following a matched call. Hence, we use a transition from $\Delta_{mr} = \{((q, A), r, q', q_\bot) \mid (q, r, A, q') \in \Delta\} \cup \{((q, A, q_G), r, (q', q_G), q_\bot) \mid (q, r, A, q') \in \Delta\}$ in order to process that matched return. We verify that we have indeed guessed the correct state by using transitions from $\Delta_v = \{((q, q), \bot, q_\bot, q_\bot) \mid q \in Q\}$. Finally, we define $\Delta_s = \{(q_\bot, \bot, q_\bot, q_\bot)\}$ to continue the run of $\mathfrak{T}'_{\mathcal{S}}$ upon encountering the sink state $q_\bot$.

Using the intuition given above, it can easily be verified that $\mathcal{L}(\mathfrak{T}'_{\mathcal{S}}) \cap \mathrm{st}(\Sigma^\omega) = \mathrm{st}(\mathit{traces}(\mathcal{S}))$ indeed holds true. Thus, we obtain the automaton $\mathfrak{T}_{\mathcal{S},\neg\varphi}$ with the properties given in the statement of this lemma as argued above. ◀

Due to Theorem 6, we obtain a novel asymptotically optimal algorithm for VLDL model checking: Given a VPS $\mathcal{S}$ and a VLDL formula φ, we construct $\mathfrak{T}$ such that $\mathcal{L}(\mathfrak{T}) = \mathrm{st}(\mathit{traces}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi))$. Recall that $\mathit{traces}(\mathcal{S}) \subseteq \mathcal{L}(\varphi)$ if and only if $\mathit{traces}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi) = \emptyset$. Since $\mathrm{st}(\mathit{traces}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi))$ is empty if and only if $\mathit{traces}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi)$ is empty, we obtain that $\mathcal{L}(\mathfrak{T})$ is empty if and only if $\mathit{traces}(\mathcal{S}) \subseteq \mathcal{L}(\varphi)$.

The automaton $\mathfrak{T}$ can be constructed in exponential time and is of exponential size in $|\varphi|$ and of polynomial size in both $|Q|$ and $|\Gamma|$. Hence, we can check $\mathfrak{T}$ for emptiness in exponential time in $|\varphi|$ and in polynomial time in both $|Q|$ and $|\Gamma|$. Since the problem of VLDL model checking is ExpTime-complete [24], this algorithm is asymptotically optimal.

6 Conclusion

In this work we have presented a correspondence between infinite words over a pushdown alphabet and infinite binary trees. Moreover, we demonstrated a construction translating VLDL formulas into tree automata that are language-equivalent with respect to the above correspondence. This construction yields novel algorithms for satisfiability and model checking of VLDL formulas that reduce the problem to the emptiness problem for tree automata. Thus, this construction leverages the strong connection between visibly pushdown languages and regular tree languages that was already exhibited by Alur and Madhusudan in their seminal

work on the former family of languages [2]. Moreover, the construction demonstrates that the well-known breakpoint construction by Miyano and Hayashi [14], which is routinely used to remove alternation from stack-free automata, can easily be adapted to transform alternating automata over visibly pushdown words into corresponding alternation-free automata over trees representing such words.

In future work, we plan to empirically evaluate and compare the algorithms presented in this work as well as those presented in earlier work [24], which reduce the satisfiability and model checking problems for VLDL to the emptiness problem for visibly pushdown automata. Recall that our novel algorithm reduces both problems to the emptiness problem for tree automata, which in turn reduces to the problem of solving a two-player Büchi game. The latter problem is well-studied due to its important applications, e.g., in program verification [1, 22] and program synthesis [12]. Hence, there exist efficient algorithms [17] for solving them as well as mature solvers [9, 10]. Thus, we expect our novel algorithm to outperform the existing approach [24] to the above problems.

Moreover, in previous work we investigated the problem of solving two-player games on a visibly pushdown arena in which the winning condition is given by a VLDL formula and determined this problem to be 3ExpTime-complete [24]. We showed membership of this problem in 3ExpTime by reducing it to the problem of solving visibly pushdown games against a winning condition given by visibly pushdown automata. Currently, we are investigating whether the approach presented in this work can be lifted to the setting of games.

Acknowledgments. The author would like to thank Martin Zimmermann for multiple fruitful discussions.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

3 Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN Model Checking and Software Verification, 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000, Proceedings*, volume 1885 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2000. doi:10.1007/10722468_7.

4 Thomas Ball and Sriram K. Rajamani. Bebop: a path-sensitive interprocedural dataflow engine. In John Field and Gregor Snelting, editors, *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering, PASTE'01, Snowbird, Utah, USA, June 18-19, 2001*, pages 97–103. ACM, 2001. doi:10.1145/379605.379690.

5 Laura Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 476–491. Springer, 2007. doi:10.1007/978-3-540-74407-8_32.

6 Laura Bozzelli and César Sánchez. Visibly Linear Temporal Logic. *J. Aut. Reas.*, 2018. In Press. doi:10.1007/s10817-017-9410-z.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Algorithms for büchi games. *CoRR*, abs/0805.2620, 2008. arXiv:0805.2620.

8 Nathanaël Fijalkow, Sophie Pinchinat, and Olivier Serre. Emptiness of alternating tree automata using games with imperfect information. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 299–311. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.299.

9 Oliver Friedmann and Martin Lange. The PGSolver Collection of Parity Game Solvers. *University of Munich*, 2009. Available at https://github.com/tcsprojects/pgsolver/blob/b88e86e31f2fe02ebcabdccd51ee73f2692ac884/doc/pgsolver.pdf.

10 Jeroen Keiren. An experimental study of algorithms and optimisations for parity games, with an application to Boolean Equation Systems. Master's thesis, Eindhoven University of Technology, 2009. Available at http://www.jeroenkeiren.nl/publications.

11 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 224–233. ACM, 1998. doi:10.1145/276698.276748.

12 Orna Kupferman and Moshe Y. Vardi. From linear time to branching time. *ACM Trans. Comput. Log.*, 6(2):273–294, 2005. doi:10.1145/1055686.1055689.

13 Christof Löding, Parthasarathy Madhusudan, and Olivier Serre. Visibly Pushdown Games. In L. Lodaya and M. Mahajan, editors, *FSTTCS 2004*, volume 3328 of *LNCS*, pages 408–420. Springer, 2005. doi:10.1007/978-3-540-30538-5_34.

14 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984. doi:10.1016/0304-3975(84)90049-5.

15 Frank Nießner. Nondeterministic tree automata. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2001. doi:10.1007/3-540-36387-4_8.

16 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

17 Yuval Rabani, editor. *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. SIAM, 2012. doi:10.1137/1.9781611973099.

18 Stefan Schwoon. *Model checking pushdown systems*. PhD thesis, Technical University Munich, Germany, 2002. URL: http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/schwoon.html.

19 Dejvuth Suwimonteerabuth, Stefan Schwoon, and Javier Esparza. jmoped: A java bytecode checker based on moped. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 541–545. Springer, 2005. doi:10.1007/978-3-540-31980-1_35.

20 Wolfgang Thomas. Automata on Infinite Objects. *Handbook of theoretical computer science, Volume B*, pages 133–191, 1990.

21 Wolfgang Thomas. Languages, Automata, and Logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.

22 Moshe Y. Vardi. Automata-theoretic model checking revisited. In Hana Chockler and Alan J. Hu, editors, *Hardware and Software: Verification and Testing, 4th International Haifa Verification Conference, HVC 2008, Haifa, Israel, October 27-30, 2008. Proceedings*, volume 5394 of *Lecture Notes in Computer Science*, page 2. Springer, 2008. doi:10.1007/978-3-642-01702-5_2.

23 Alexander Weinert. VLDL satisfiability and model checking via tree automata. *CoRR*, abs/1708.00699, 2017. arXiv:1708.00699.

24 Alexander Weinert and Martin Zimmermann. Visibly linear dynamic logic. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65 of *LIPIcs*, pages 28:1–28:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.28.

25 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.