10th International Conference on Fun with Algorithms

FUN 2021, May 30–June 1, 2021, Favignana Island, Sicily, Italy

Edited by

Martin Farach-Colton
Giuseppe Prencipe
Ryuhei Uehara

LIPICS

Editors

Martin Farach-Colton
Rutgers University, NJ, USA
martin@farach-colton.com

Giuseppe Prencipe
Università di Pisa, Italy
giuseppe.prencipe@unipi.it

Ryuhei Uehara
Japan Advanced Institute of Science and Technology, Ishikawa, Japan
uehara@jaist.ac.jp

ACM Classification 2012
Theory of computation; Mathematics of computing; Computing methodologies; Security and privacy; Applied computing; Information systems

ISBN 978-3-95977-145-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-145-0.

Publication date
September, 2020

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 https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.FUN.2021.0

ISBN 978-3-95977-145-0 **ISSN 1868-8969** **https://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

https://www.dagstuhl.de/lipics

Contents

Regular Papers

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

FUN with Algorithms is dedicated to the use, design, and analysis of algorithms and data structures, focusing on results that provide amusing, witty but nonetheless original and scientifically profound contributions to the area. Donald Knuth's famous quote captures this spirit nicely:*.... pleasure has probably been the main goal all along. But I hesitate to admit it, because computer scientists want to maintain their image as hard-working individuals who deserve high salaries. Sooner or later society will realize that certain kinds of hard work are in fact admirable even though they are more fun than just about anything else.*

The previous FUNs were held in Elba Island, Italy; in Castiglioncello, Tuscany, Italy; in Ischia Island, Italy; in San Servolo Island, Venice, Italy; in Lipari Island, Sicily, Italy; and in La Maddalena Island, Sardinia, Italy. Special issues of Theoretical Computer Science, Discrete Applied Mathematics, and Theory of Computing Systems were dedicated to them.

This volume contains the papers that will be presented at the 10th International Conference on Fun with Algorithms 2020. The conference was originally scheduled to be at Island of Favignana, Sicily, Italy, from the 8th to the 10th of June, 2020. Unfortunately, due to the COVID-19 global emergency, we were forced to delay it, which was not so fun. Fortunately, FUN is held every two years; thus, we planned to postpone it until summer 2021. Different dates, same venue, same FUN!

The call for papers attracted 48 submissions from all over the world, addressing a wide variety of topics, reviewed by 31 Program Committee members. After a careful reviewing process and a thorough discussion, the committee decided to accept 24 papers. Extended versions of selected papers will appear in a special issue. In addition, the program features invited talks by Eva Rotenberg and Bettina Speckmann.

We thank all authors who submitted their work to FUN 2020/2021, all Program Committee members for their expert assessments and the ensuing discussions, all external reviewers for their kind help, and Atsuki Nagao for taking care of the web management of the conference. We used EasyChair (http://www.easychair.org/), that greatly facilitated the entire preparation of the conference, for handling submissions, reviews, the selection of papers, and the production of this volume. Warm thanks also go to Michael Wagner, and to the LIPIcs team, for following carefully the process of proceedings' publication in LIPIcs series.

When we are editing this volume, the things are not yet settled. However, we hope we can share fun with algorithms even if, at least for now, it may be only shared by online.

May, 2020

Martin Farach-Colton
Giuseppe Prencipe
Ryuhei Uehara

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Committee

Flavia Bonomo, U. of Buenos Aires, Argentina
Arnaud Casteigts, U. de Bordeaux, France
Graham Cormode, U. of Warwick, United Kingdom
Martin Farach-Colton, Rutgers U., USA (co-chair)
Paolo Ferragina, U. di Pisa, Italy
Paola Flocchini, U. of Ottawa, Canada
Jie Gao, Rutgers U., USA
Seth Gilbert, National U. of Singapore, Singapore
Inge Li Gørtz , Danmarks Tekniske U., Denmark
John Iacono, U. Libre de Bruxelles, Belgium
Hiro Ito, U. of Electro-Communications, Japan
Chuzo Iwamoto, Hiroshima U., Japan
Tomasz Jurdzinski, U. Wroc?awski, Poland
Irina Kostitsyna, Technische U. Eindhoven, Nederlands
Michal Koucky, U. Karlova, Czech Republic
Stefan Langerman, U. Libre de Bruxelles, Belgium
Giuseppe Di Luna, Sapienza U. di Roma, Italy
Javier Marenco, U. Nacional de General Sarmiento, Argentina
Miguel A. Mosteiro, Pace U., USA
Yoshio Okamoto, U. of Electro-Communications, Japan
Solon Pissis, Centrum Wiskunde & Informatica, Nederlands
Simon Puglisi, Helsingin yliopisto, Finland
Eva Rotenberg, Danmarks Tekniske U., Denmark
Shikha Singh, Williams College, USA
Akira Suzuki, Tohoku U., Japan
Ryuhei Uehara, JAIST, Japan (co-chair)
Yushi Uno, Osaka Prefecture U., Japan
Przemek Uznanski, U. Wrocławski, Poland
Giovanni Viglietta, JAIST, Japan
Prudence Wong, U. of Liverpool, United Kingdom
Maxwell Young, Mississippi State U., USA

Steering Commitee

Hiro Ito, U. of Electro-Communications, Japan
Stefano Leonardi, Sapienza U. di Roma, Italy
Linda Pagli, U. di Pisa, Italy
Giuseppe Prencipe, U. di Pisa, Italy
Geppino Pucci, U. degli Studi di Padova , Italy
Nicola Santoro, Carleton U., Canada

Organizers

Linda Pagli, U. di Pisa, Italy
Giuseppe Prencipe, U. di Pisa, Italy
Atsuki Nagao, Ochanomizu U., Japan (web manager)
Giovanni Viglietta, JAIST, Japan (web manager)
Miguel A. Mosteiro, Pace U., USA (publicity chair)

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Adam Hesterberg
achesterberg@gmail.com
United States
Massachusetts Institute of Technology

Adam Suhl
asuhl@mit.edu
United States
Algorand

Adele Rescigno
arescigno@unisa.it
Italy
University of Salerno

Aiden Calvert
calverta20@themsms.org
United States
Mississippi School for Math. and Science

Alex Churchill
alex.churchill@cantab.net
United Kingdom
Independent

Alexander Koch
alexander.koch@kit.edu
Germany
Karlsruhe Institute of Technology

Antonio Molina Lovett
antonio@amolina.ca
Canada
University of Waterloo

Aris Anagnostopoulos
aris@diag.uniroma1.it
Italy
Sapienza University of Rome

Aristides Gionis
aristides.gionis@aalto.fi
Finland
Aalto University

Atsuki Nagao
a-nagao@is.ocha.ac.jp
Japan
Ochanomizu University

Austin Herrick
aherrick@wharton.upenn.edu
United States
University of Pennsylvania

Avi Zeff
avizeff@mit.edu
United States
MIT CSAIL

Aviv Adler
adlera@mit.edu
United States
Massachusetts Institute of Technology

Bernardo Subercaseaux
bernardosubercaseaux@gmail.com
Chile
Universidad de Chile

Charles Guinn
cguinn@princeton.edu
United States
Princeton University

Daiki Miyahara
daiki.miyahara.q4@dc.tohoku.ac.jp
Japan
Tohoku University

Daniel Frishberg
dfrishbe@uci.edu
United States
University of California, Irvine

David Eppstein
eppstein@uci.edu
United States
University of California, Irvine

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

David Wehner
david.wehner@inf.ethz.ch
Switzerland
ETH Zurich

Davide Bilò
davidebilo@uniss.it
Italy
University of Sassari, Italy

Dylan Hendrickson
dylanhen@mit.edu
United States
Massachusetts Institute of Technology

Dylan Hendrickson
dylanhen@mit.edu
United States
Massachusetts Institute of Technology

Erik D. Demaine
edemaine@mit.edu
United States
Massachusetts Institute of Technology

Eryk Kopczynski
erykk@duch.mimuw.edu.pl
Poland
University of Warsaw

Fabian Frei
fafrei@inf.ethz.ch
Switzerland
ETH Zurich

G.W. van der Heijden
g.w.v.d.heijden@student.tue.nl
Netherlands
TU Eindhoven

Gennaro Cordasco
gennaro.cordasco@unicampania.it
Italy
University of Campania "Luigi Vanvitelli"

Giacomo Scornavacca
giacomo.scornavacca@graduate.univaq.it
Italy
University of Sassari, Italy

Guido Proietti
guido.proietti@univaq.it
Italy
Università L'Aquila, Italy and Istituto di Analisi dei Sistemi ed Informatica, IASI-CNR, Roma, Italy

Guilherme da Fonseca
guilherme.fonseca@lis-lab.fr
France
Université Aix Marseille

Hideaki Sone
Japan
Tohoku University

Irina Kostitsyna
i.kostitsyna@tue.nl
Netherlands
TU Eindhoven

Jayson Lynch
jaysonl@mit.edu
United States
Massachusetts Institute of Technology

Jeffrey Bosboom
jbosboom@csail.mit.edu
United States
Massachusetts Institute of Technology

Jeffrey Shallit
shallit@uwaterloo.ca
Canada
University of Waterloo

Jérémy Barbay
jeremy@barbay.cl
Chile
Universidad de Chile

Josh Brunner
brunnerj@mit.edu
United States
Massachusetts Institute of Technology

Joshua Ani
joshuaa@mit.edu
United States
Massachusetts Institute of Technology

Juan Jose Besa Vial
jjbesavi@uci.edu
United States
University of California, Irvine

Julian Wellman
wellman@mit.edu
United States
Massachusetts Institute of Technology

Kazumasa Shinagawa
shinagawakazumasa@gmail.com
Japan
Tokyo Institute of Technology

Leo Robert
leo.robert@uca.fr
France
LIMOS, University Clermont Auvergne

Leon Witzman
lwitzman@edu.uwaterloo.ca
Canada
University of Waterloo

Lily Chung
ikdc@mit.edu
United States
Massachusetts Institute of Technology

Lloyd E. Lo-Wong
l.e.lo-wong@student.tue.nl
Netherlands
TU Eindhoven

Loïc Crombez
loic.crombez@uca.fr
France
Université Clermont Auvergne

Luciano Gualà
guala@mat.uniroma2.it
Italy
Dipartimento di Matematica
Università di Tor Vergata, Roma, Italy

Luisa Gargano
lgargano@unisa.it
Italy
University of Salerno

Martha Osegueda
mosegued@uci.edu
United States
University of California, Irvine

Martin L. Demaine
mdemaine@mit.edu
United States
Massachusetts Institute of Technology

Maxwell Young
my325@msstate.edu
United States
Mississippi State University

Nikos Parotsidis
nickparo1@gmail.com
Italy
Google

Nil Mamano
nmamano@uci.edu
United States
University of California, Irvine

Pascal Lafourcade
pascal.lafourcade@udamail.fr
France
LIMOS, University Clermont Auvergne

Peter Rossmanith
rossmani@cs.rwth-aachen.de
Germany
RWTH Aachen

Qian Zhou
qz70@msstate.edu
United States
Mississippi State University

Quanquan C. Liu
quanquan@mit.edu
United States
Massachusetts Institute of Technology

Quentin Bramas
quentin.bramas@gmail.com
France
ICUBE, Université de Strasbourg, CNRS

Remco J. A. Surtel
r.j.a.surtel@student.tue.nl
Netherlands
TU Eindhoven

Ross Dempsey
rossd97@gmail.com
United States
Princeton University

So Takeshige
so.takeshige.q1@dc.tohoku.ac.jp
Japan
Tohoku University

Stefan Walzer
stefan.walzer@tu-ilmenau.de
Germany
TU Ilmenau

Stefano Leucci
stefano.leucci@univaq.it
Italy
University of L'Aquila

Stella Biderman
stellabiderman@gmail.com
United States
Georgia Institute of Technology

Stéphane Devismes
stephane.devismes@univ-grenoble-alpes.fr
France
VERIMAG UMR 5104

Suthee Ruangwises
ruangwises.s.aa@m.titech.ac.jp
Japan
Tokyo Institute of Technology

Takaaki Mizuki
tm-paper+zerotate@
g-mail.tohoku-university.jp
Japan
Tohoku University

Thomas Brocken
t.brocken@student.tue.nl
Netherlands
TU Eindhoven

Thomas Lidbetter
tflidbetter@mta.ca
Canada
University of Waterloo

Timothy Johnson
tujohnso@uci.edu
United States
University of California, Irvine

Tomasz Idziaszek
idziaszek@mimuw.edu.pl
Poland
algonotes.com

Toshiya Itoh
titoh@c.titech.ac.jp
Japan
Tokyo Institute of Technology

Trevor Clokie
trevor.clokie@uwaterloo.ca
Canada
University of Waterloo

William Kuszmaul
kuszmaul@mit.edu
United States
Massachusetts Institute of Technology

William Maxwell
maxwellw@oregonstate.edu
United States
Oregon State University

Yan Gerard
yan.gerard@uca.fr
France
Université Clermont Auvergne

Yevhenii Diomidov
diomidov@mit.edu
United States
Massachusetts Institute of Technology

Tatamibari Is NP-Complete

Aviv Adler
Massachusetts Institute of Technology, Cambridge, MA, USA
adlera@mit.edu

Jeffrey Bosboom
Massachusetts Institute of Technology, Cambridge, MA, USA
jbosboom@mit.edu

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA
edemaine@mit.edu

Martin L. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA
mdemaine@mit.edu

Quanquan C. Liu
Massachusetts Institute of Technology, Cambridge, MA, USA
quanquan@mit.edu

Jayson Lynch
Massachusetts Institute of Technology, Cambridge, MA, USA
jaysonl@mit.edu

Abstract

In the Nikoli pencil-and-paper game Tatamibari, a puzzle consists of an $m \times n$ grid of cells, where each cell possibly contains a clue among ⊞, ⊟, ◫. The goal is to partition the grid into disjoint rectangles, where every rectangle contains exactly one clue, rectangles containing ⊞ are square, rectangles containing ⊟ are strictly longer horizontally than vertically, rectangles containing ◫ are strictly longer vertically than horizontally, and no four rectangles share a corner. We prove this puzzle NP-complete, establishing a Nikoli gap of 16 years. Along the way, we introduce a gadget framework for proving hardness of similar puzzles involving area coverage, and show that it applies to an existing NP-hardness proof for Spiral Galaxies. We also present a mathematical puzzle font for Tatamibari.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Nikoli puzzles, NP-hardness, rectangle covering

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.1

Related Version This paper is also available on arXiv at `https://arXiv.org/abs/2003.08331`.

Supplementary Material The Z3-based Tatamibari solver and figures from Tatamibari NP-hardness paper are available at `https://github.com/jbosboom/tatamibari-solver` [4].

Acknowledgements This work was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.890) taught by Erik Demaine in Fall 2014. We thank the other participants of that class for related discussions and providing an inspiring atmosphere.

1 Introduction

Nikoli is perhaps the world leading publisher of pencil-and-paper logic puzzles, having invented and/or popularized hundreds of different puzzles through their *Puzzle Communication Nikoli* magazine and hundreds of books. Their English website [29] currently lists 38 puzzle types, while their "omopa list" [28] currently lists 456 puzzle types and their corresponding first appearance in the magazine.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 1; pp. 1:1–1:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Nikoli's puzzles have drawn extensive interest by theoretical computer scientists (including the FUN community): whenever a new puzzle type gets released, researchers tackle its computational complexity. For example, the following puzzles are all NP-complete: Bag / Corral [13], Country Road [20], Fillomino [31], Hashiwokakero [8], Heyawake [19], Hiroimono / Goishi Hiroi [7], Hitori [17, Section 9.2], Kakuro / Cross Sum [32], Kurodoko [22], Light Up / Akari [25], LITS [26], Masyu / Pearl [14], Nonogram / Paint By Numbers [30], Numberlink [23, 2], Nurikabe [24, 18], Shakashaka [12, 3], Slitherlink [32, 31, 1], Spiral Galaxies / Tentai Show [15], Sudoku [32, 31], Yajilin [20], and Yosenabe [21].

Allen et al. [5] defined the ***Nikoli gap*** to be the amount of time between the first publication of a Nikoli puzzle and a hardness result for that puzzle type. They observed that, while early Nikoli puzzles have a gap of 10–20 years, puzzles released within the past ten years tend to have a gap of < 5 years.

In this paper, we prove NP-completeness of a Nikoli puzzle first published in 2004 [27] (according to [28]), establishing a Nikoli gap of 16 years.[1] Specifically, we prove NP-completeness of the Nikoli puzzle ***Tatamibari*** (タタミバリ), named after Japanese tatami mats. A Tatamibari ***puzzle*** consists of a rectangular $m \times n$ grid of unit-square cells, some k of which contain one of three different kinds of clues: ⊞, ◫, and ⊟. (The remaining $m \cdot n - k$ cells are empty, i.e., contain no clue.) A ***solution*** to such a puzzle is a set of k grid-aligned rectangles satisfying the following constraints:

1. The rectangles are disjoint.
2. The rectangles together cover all cells of the puzzle.
3. Each rectangle contains exactly one symbol in it.
4. The rectangle containing a ⊞ ("square") symbol is a square, i.e., has equal width (horizontal dimension) and height (vertical dimension).
5. The rectangle containing a ⊟ ("horizontal") symbol has greater width than height.
6. The rectangle containing a ◫ ("vertical") symbol has greater height than width.
7. No four rectangles share the same corner (***four-corner constraint***).

To prove our hardness result, we first introduce in Section 2 a general "gadget area hardness framework" for arguing about (assemblies of) local gadgets whose logical behavior is characterized by area coverage. Then we apply this framework to prove Tatamibari NP-hard in Section 3. In Appendix A, we show that our framework applies to at least one existing NP-hardness proof, for the Nikoli puzzle Spiral Galaxies [15].

We also present in Section 4 a mathematical puzzle font [11] for Tatamibari, consisting of 26 Tatamibari puzzles whose solutions draw each letter of the alphabet. This font enables writing secret messages, such as the one in Figure 1, that can be decoded by solving the Tatamibari puzzles. This font complements a similar font for another Nikoli puzzle, Spiral Galaxies [6].

2 Gadget Area Hardness Framework

Puzzles. The ***gadget area hardness framework*** applies to a general ***puzzle type*** (e.g., Tatamibari or Spiral Galaxies) that defines puzzle-specific mechanics. In general, a ***subpuzzle*** is defined by an embedded planar graph, whose finite faces are called ***cells***, together with an optional ***clue*** (e.g., number or symbol) in each cell. The puzzle type defines which subpuzzles are valid ***puzzles***, in particular, which clue types and planar graphs are permitted, as well as any additional **properties** guaranteed by a hardness reduction producing the puzzles.

[1] While this gap may be caused by the puzzle being difficult to prove hard or simply overlooked (or both), we can confirm that it took us nearly six years to write this paper.

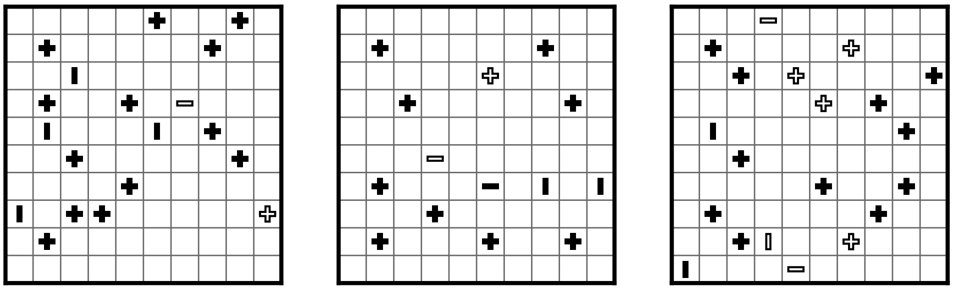


Figure 1 What do these Tatamibari puzzles spell when solved and the dark clues' rectangles are filled in? Figure 14 gives a solution.

We will use the unrestricted notion of subpuzzles to define gadgets. Define an ***area*** of a puzzle to be a connected set of cells. An ***instance*** of a subpuzzle in a puzzle is an area of the puzzle such that the restriction of the puzzle to that area (discarding all cells and clues outside the area) is exactly the subpuzzle.

Solutions. An ***area assignment*** (potential solution) for a (sub)puzzle is a mapping from clues to areas such that (1) the areas disjointly partition the cells of the (sub)puzzle, and (2) each area contains the cell with the corresponding clue. The puzzle type defines when an area assignment is an actual ***solution*** to a puzzle or a ***local solution*** to a subpuzzle.

Gadgets. A ***gadget*** is a subpuzzle plus a partition of its ***entire*** area (all of its cells) into one ***mandatory*** area and two or more ***optional*** areas, where all clues are in the mandatory area. A hardness reduction using this framework should compose puzzles from instances of gadgets that overlap only in optional areas, and provide a ***filling algorithm*** that defines which clues are in the areas exterior to all gadgets. Each gadget thus defines the entire set of clues of the puzzle within the gadget's (mandatory) area.

For a given gadget, a ***gadget area assignment*** is an area assignment for the subpuzzle that satisfies three additional properties:

1. the assigned areas cover the gadget's mandatory area;
2. every optional area is either fully covered or fully uncovered by assigned areas; and
3. every assigned area lies within the gadget's entire area.

A ***gadget solution*** is a gadget area assignment that is a local solution as defined by the puzzle type.

Profiles. A ***profile*** of a gadget is a subset of the gadget's entire area. A profile is ***proper*** if it satisfies two additional properties:

1. the profile contains the mandatory area of the gadget; and
2. every optional area of the gadget is either fully contained or disjoint from the profile.

Every gadget area assignment induces a proper profile, namely, the union of the assigned areas.

A profile is ***locally solvable*** if there is a gadget solution with that profile. A profile is ***locally impossible*** if, in any puzzle containing an instance of the gadget, there is no solution to the entire puzzle such that the union of the areas assigned to the clues of the gadget instance is that profile. These notions might not be negations of each other because of differences between local solutions of a subpuzzle and solutions of a puzzle.

Each gadget is characterized by a ***profile table*** (like a truth table) that lists all profiles that are locally solvable, and for each such profile, gives a gadget solution. A profile table is ***proper*** if it contains only proper profiles. A profile table is ***complete*** if every profile not in the table is locally impossible. A hardness reduction using this framework should prove that the profile table of each gadget is proper and complete, in particular, that any improper profile is locally impossible.

Given a puzzle containing some gadget instances, a ***profile assignment*** specifies a profile for each gadget such that the profiles are pairwise disjoint and the union of the profiles covers the union of the entire areas of the gadgets. In particular, such an assignment decides which overlapping optional areas are covered by which gadgets. A profile assignment is ***valid*** if every gadget is locally solvable with its assigned profile, i.e., every assigned profile is in the profile table of the corresponding gadget.

A hardness reduction using this framework should prove that every valid profile assignment can be extended to a solution of the entire puzzle by giving a ***composition algorithm*** for composing local solutions from the profile tables of the gadgets, possibly modifying these local solutions to be globally consistent, and extending these solutions to assign areas to clues from the filling algorithm (exterior to all gadgets).

3 Tatamibari is NP-hard

In this section, we prove Tatamibari NP-hard by a reduction from planar rectilinear monotone 3SAT. In Section 3.1 we briefly discuss a more constrained (but still NP-hard) variant of the classic 3SAT problem from which we will make our reduction; in Section 3.2, Section 3.3, and Section 3.4, we describe the gadgets (wires, variables, and clauses) from which we build the reduction; in Section 3.5, we discuss how the spaces between the gadgets are filled; and in Section 3.6 we use everything to show the main result.

3.1 Reduction Overview

We reduce from *planar rectilinear monotone 3SAT*, proved NP-hard in [9]. An instance of planar rectilinear monotone 3SAT comes with a planar rectilinear drawing of the clause-variable graph in which each variable is a horizontal segment on the x-axis and each clause is a horizontal segment above or below the axis, with rectilinear edges connecting variables to the clauses in which they appear. Each clause contains only positive or negative literals (i.e., is monotone); clauses containing positive (negative) literals appear above (below) the variables. We can always lengthen the variable and clause segments to remove bends in the edges, so we assume the edges are vertical line segments. We can further assume that each clause consists of exactly three (not necessarily distinct) literals: if a clause has $k < 3$ literals, we can just duplicate one of the clauses $3 - k$ times, which is easy to do while preserving the tri-legged rectilinear layout.

We create and arrange our gadgets directly following the drawing, possibly after scaling it up; see Figure 2. Edges between variables and clauses are represented by *wire gadgets* that communicate a truth value in the parity of their covering. For each variable, we create a *variable gadget*, which is essentially a block of wires forced to have the same value, and place it to fill the variable's line segment in the drawing. For each clause, we create a *clause gadget* with three wire connection points and place it to fill the clause's line segment. Negative clauses and wires representing negative literals are mirrored vertically. Both the variable and clause gadgets can telescope to any width to match the drawing; unused wires from the variable gadgets are terminated at a *terminator*. By our assumption that the edges are vertical segments, we do not need a turn gadget.

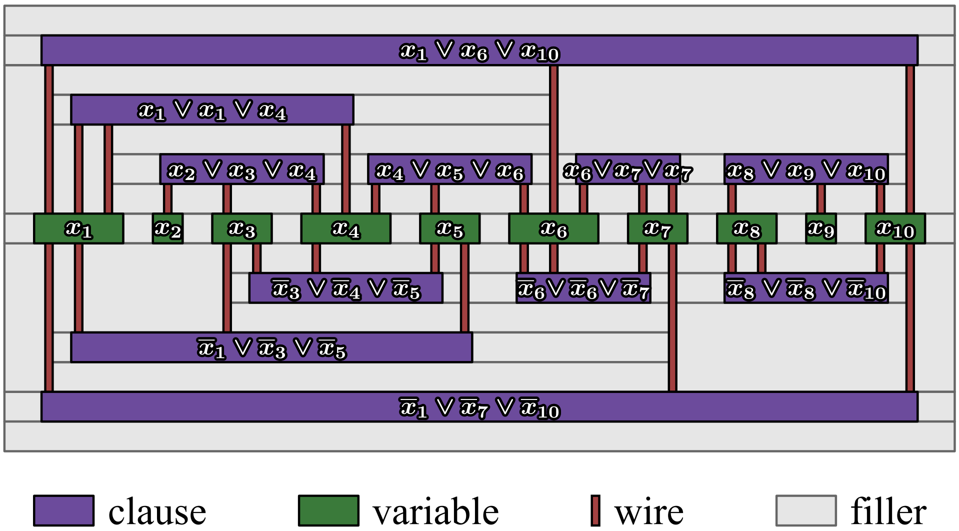

$x_1 \vee x_6 \vee x_{10}$
$x_1 \vee x_1 \vee x_4$
$x_2 \vee x_3 \vee x_4$
$x_4 \vee x_5 \vee x_6$
$x_6 \vee x_7 \vee x_7$
$x_8 \vee x_9 \vee x_{10}$
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
$\overline{x}_3 \vee \overline{x}_4 \vee \overline{x}_5$
$\overline{x}_6 \vee \overline{x}_6 \vee \overline{x}_7$
$\overline{x}_8 \vee \overline{x}_8 \vee \overline{x}_{10}$
$\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_5$
$\overline{x}_1 \vee \overline{x}_7 \vee \overline{x}_{10}$
clause
variable
wire
filler

Figure 2 The overall layout of the Tatamibari puzzles produced by our reduction follows the input planar rectilinear monotone 3SAT instance. Clause, variable, and wire gadgets are represented by purple, green, and red rectangles. Not drawn are terminator gadgets at the base of all unused copies of variables. Grey rectangles correspond to individual filler clues. Figure inspired by [9, Figure 2].

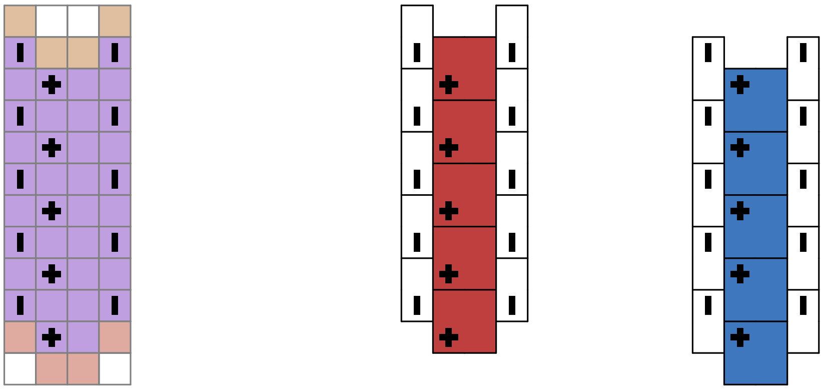
Covering a clause gadget without double-covering or committing a four-corner violation requires at least one of its attached wires to be covered with the satisfying parity (the true parity for positive clauses and the false parity for negative clauses).

To ensure clues in one gadget do not interfere with other gadgets, the wire gadget is surrounded on its left and right sides by sheathing of ◫ clue rectangles and the clause gadget is surrounded on three sides by a line of ⊞ clues forced to form 1 × 1 rectangles. Wire sheathing also ensures neighboring wires do not constrain each other, except in variable gadgets where the sheathing is deliberately punctured.

In our construction, gadgets will not overlap in their mandatory areas, so in the intended solutions, the mandatory area will be fully covered by rectangles satisfying the gadget's clues. Also in our construction, every optional area will belong to exactly two gadgets, and in the intended solutions, such an area will be covered by clues in exactly one of those gadgets.

To apply the gadget area hardness framework, we define a ***local solution*** of a subpuzzle to be a disjoint set of rectangles satisfying the gadget's clues and the property that no four of these rectangles share a corner. (At the boundary of the subpuzzle, there is no constraint.) Our composition algorithm will combine these local solutions by staggering rectangles to avoid four-corner violations on the boundary of and exterior to gadgets. We will prove that valid profile assignments correspond one-to-one to satisfying truth assignments of the 3SAT instance.

We developed our gadgets using a Tatamibari solver based on the SMT solver Z3 [10]. The solver and machine-readable gadget diagrams are available [4]. Unfortunately, the solver can only verify the correctness of constant-size instances of the gadgets, but the variable and clause gadgets must telescope to arbitrary width. Thus we still need to give manual proofs of correctness.



(a) An unsolved wire gadget. Mandatory area is purple and optional areas are brown.

(b) Wire communicating false.

(c) Wire communicating true.

Figure 3 Wire gadget and its profile table. The wire can be extended to arbitrary height by repeating rows. Note that between figures (b) and (c), the clues stay in the same place (and the rectangles shift to represent the different values of the wire).

3.2 Wire Gadgets and Terminators

The wire gadget, shown in Figure 3, consists of a column of ⊞ clues surrounded by ◫ clues which encodes a truth value in the parity of whether the squares are oriented with the ⊞ clues in their upper left or lower left corners. We will call this the *wire parity* or *wire value*. In this construction, only vertical wires are needed, and thus we do not give a turn gadget or horizontal wire. We call the column containing the ⊞ clues and the empty column next to it the *inner wire*. The inner wire is covered by columns of alternating ◫ clues, called the *(inner) sheathing*. In the overall reduction, ◫ clues in columns just outside the wire at its ends (in the variable gadget and either the clause or terminator gadget) add a further layer of sheathing (called the *outer sheathing*) outside the wire gadget, ensuring neighboring wires do not constrain each other.

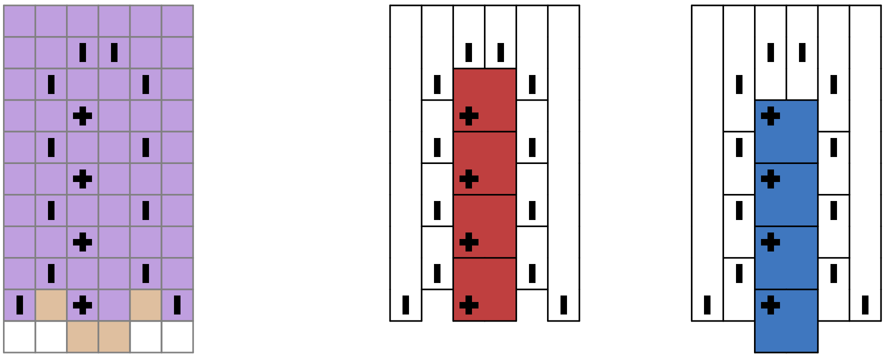
The following lemmas will show that the ⊞ clues in the wire must be covered by 2×2 squares, the squares must all have the same parity, and the wire does not impart any significant constraints onto the surrounding region. These lemmas assume that no rectangle from a ◫ clue can reach the cells to the right of the top and bottom ⊞ clues in the wire, a property which we call ***safe placement***. We discharge this assumption in Section 3.5 by showing all wire gadgets produced by our reduction are safely placed.

▶ **Lemma 3.1.** *Each* ⊞ *in a safely placed wire covers a* 2×2 *square in the wire.*

Proof. There is no 3×3 square in the wire that contains a ⊞ clue but does not contain any other clue. Thus we cannot cover the ⊞ clue by squares larger than 2×2.

Now suppose we cover a ⊞ clue by a 1×1 square. Now the cells immediately above and below this clue must be covered. The ◫ clues must be taller than they are long, so they cannot cover these cells. Thus we must cover them by squares containing the ⊞ clues above and below. This leaves the cell directly to the right of the 1×1 uncovered. It is easy to see that this cannot be covered by the nearby ⊞ clues or ◫ clues. The cells next to the top and bottom clues cannot be covered by a ◫ clue from outside the gadget by our assumption that the wire is safely placed.

The only remaining possibility is a ⊟ clue from outside the gadget extending into the wire gadget. Such a rectangle cannot extend entirely through the wire because the ◫ clues in the sheathing and the ⊞ clues inside the wire are in alternating rows. If the external



(a) An unsolved terminator gadget. Mandatory area is purple and optional area is brown. **(b)** Terminating a false wire. **(c)** Terminating a true wire.

Figure 4 Terminator gadget and its profile table.

horizontal rectangle enters the wire from the right and covers a cell next to the ⊞ clue, that ⊞ clue is forced to be a 1 × 1 rectangle and the cell above it must be covered by the next ⊞ clue above. This results in a four-corner violation involving the two ⊞ clues and the left sheathing except when the ⊞ clue is at the bottom of the wire. In that case, the external horizontal rectangle blocks the bottom-right sheathing clue, making it 1 × 1 and unsatisfied. ◀

▶ **Corollary 3.2.** *Satisfied safely placed wires must have all of their* 2×2 *squares with the* ⊞ *clues in the lower left corner or all in the upper left corner.*

Proof. By Lemma 3.1 all ⊞ clues must be covered by 2×2 squares. To change whether the ⊞ clues are in the lower left or upper left, we will end up leaving a row of two cells between clues blank. By the same arguments in Lemma 3.1 these cannot be covered by the nearby ⊞ clues or ◫ clues. ◀

▶ **Lemma 3.3.** *The* ◫ *clues making up the inner sheathing of satisfied safely placed wires must be covered by* 1×2 *rectangles of opposite parity to the wire's squares.*

Proof. By Corollary 3.2 the wire has one of two parities of squares. If a vertical rectangle ends with the same y coordinate as an adjacent square, then we will have three right angles at a single corner, forcing a four-corner violation or uncovered cell. Because the squares are 2×2, a vertical rectangle of odd height guarantees one of the ends will share a y-coordinate with one of the squares. The ◫ clues occur every other cell, so the vertical rectangles cannot be of length greater than 3. This forces them to be of length 2 and staggered with respect to the squares. ◀

▶ **Theorem 3.4.** *The safely placed wire gadget's profile table is proper and complete.*

Proof. By Lemma 3.1, each optional area must be fully covered or fully uncovered by the neighboring ⊞ clue, so the profile table is proper. Corollary 3.2 fixes the ⊞ clue parity and Lemma 3.3 fixes the sheathing parity, so all other profiles are locally impossible, so the profile table is complete. ◀

We also have a terminator gadget to terminate unused wires regardless of their parity. The terminator gadget is shown in Figure 4.

▶ **Lemma 3.5.** *The terminator does not constrain the wire parity.*

Proof. Figures 4b and 4c show solutions of the terminator with both parities. The same arguments about wire correctness show this gadget does not allow any additional wire solutions nor constrain other gadgets. ◀

▶ **Theorem 3.6.** *The terminator gadget's profile table is proper and complete.*

Proof. The profile table in Figure 4 contains only proper profiles, so the profile table is proper. By the same arguments in Lemma 3.1, the two local solutions shown are the only way to cover the wire part of the gadget. A horizontal rectangle from a ⊟ outside the gadget could cover part of the top row of the gadget (or the entire top row when terminating a true wire) while leaving the clues in the gadget satisfied and covering the remaining area. We prevent this through the global layout: all clause gadgets (the only gadget containing ⊟ clues) appear strictly above (for positive clauses) or strictly below (for negative clauses) all terminator gadgets, so it is not possible for any ⊟ rectangles to cover area in the clause gadget. Thus all other profiles are locally impossible, so the profile table is complete. ◀

3.3 Variable Gadgets

The variable gadget is essentially a series of wires placed next to each other with devices we call ***couplers*** in between. Each coupler acts essentially as an "equality" constraint between neighboring wires, thus forcing all the wires connected via a series of couplers to represent the same variable; this collection of wires then forms the ***variable gadget*** of the reduction.

Each coupler takes two columns, and consists of (i) a ⊞ clue which interacts with the inner sheathing of the wires to force equality, and (ii) eight ◫ clues (two above and two below the ⊞ clue on each column), which prevent the inner sheathings of the neighboring wires from constraining each other (except through the ⊞ clue itself). See Figure 7 for an example with two wires; additional wires can be added to either side of variable by using more couplers (see Figure 6).

First, notice that both wires are constrained to have their squares in one of two parities by the inner sheathing, as in Corollary 3.2. It is also important that wires do not constrain each other outside the couplers, either directly (if they happen to be adjacent) or indirectly (through the space in between); we address this in Section 3.5.

Now we have to show that two wires separated by the coupler must be in the same configuration. This happens because the wire parity forces the parity of the inner sheathing, which forces the parity of the coupler, which then forces the partiy of the inner sheathing and the wire parity of the next wire over.

▶ **Lemma 3.7.** *The coupler has only two valid coverings of its* ⊞ *clue.*

Proof. The location of the eight ◫ clues around the ⊞ clue ensure that it cannot be larger than 2×2. By Corollary 3.2 we know that the wire gadgets next to the coupler must have their inner sheathing as 2×1 rectangles in either the up or down position. If the ⊞ clue is covered by a 1×1 it will create a four-corner violation with the inner sheathing. Thus it must be one of the two possible positions for a 2×2 square. If both inner sheathings have the same parity, as in Figure 7 then the constraints can be locally satisfied. ◀

▶ **Lemma 3.8.** *All wires in a variable gadget must have the same value (i.e. upwards branches must have the same orientation).*

Proof. We know the coupler has at most two ways to satisfy its constraints, corresponding to a 2×2 square in either the up or down position. Notice that the inner sheathing of both wires must be of different parity from the square or they will cause a four-corner violation.

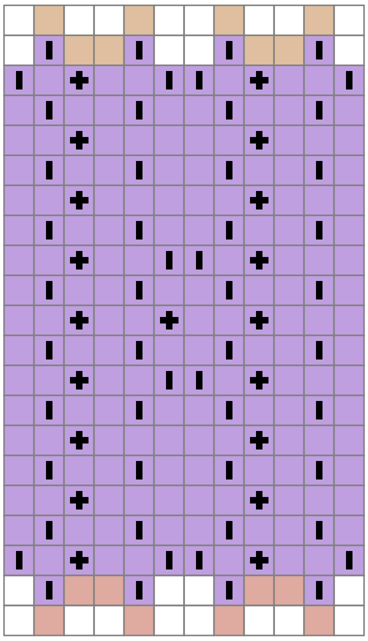


Figure 5 The variable gadget. Mandatory area is purple and optional areas are brown.

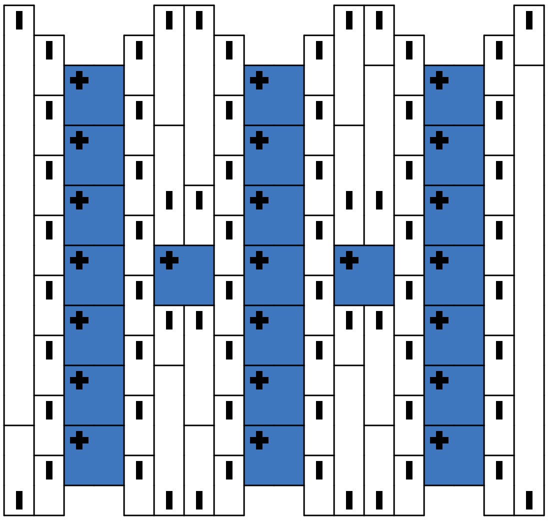


Figure 6 A variable gadget widened to provide three wires, shown here set to true.

Thus the inner sheathing must have the same parity, ensuring that the wires themselves must have the same parity. If multiple wires are all connected by couplers, then they will all be forced to have the same parity by the same local argument. ◀

▶ **Lemma 3.9.** *The variable gadget is locally solvable with a given profile if and only if the profile satisfies (i) all upwards branches have the same orientation, (ii) all downwards branches have the same orientation, and (iii) upwards and downwards branches have opposite orientations from each other.*

Proof. The "only if" direction follows from Lemma 3.8 and Corollary 3.2 (each wire individually must have opposite orientation for upwards and downwards branches due to the couplers, and all wires in the gadget must have the same upward orientation).

The "if" direction follows from Lemma 3.5, the individual solvability of each wire and terminator in both orientations (as shown in Figure 3b, Figure 3c, Figure 4b, and Figure 4c), and the solvability of the couplers given that adjacent wires have the same orientation (Figure 7). Neighboring wires (within the variable gadget) do not conflict with each other (outside of the coupler) because of the "outer sheathing" columns separating them; the meeting points of the two clues in each "outer sheathing" column can be adjusted to avoid four-corner violations with each other, as well as avoiding four-corner violations with the neighboring "inner sheathing". ◀

Note that this lemma is what we want from a variable gadget: it is locally solvable if and only if its profile corresponds to a specific value for the variable it represents.

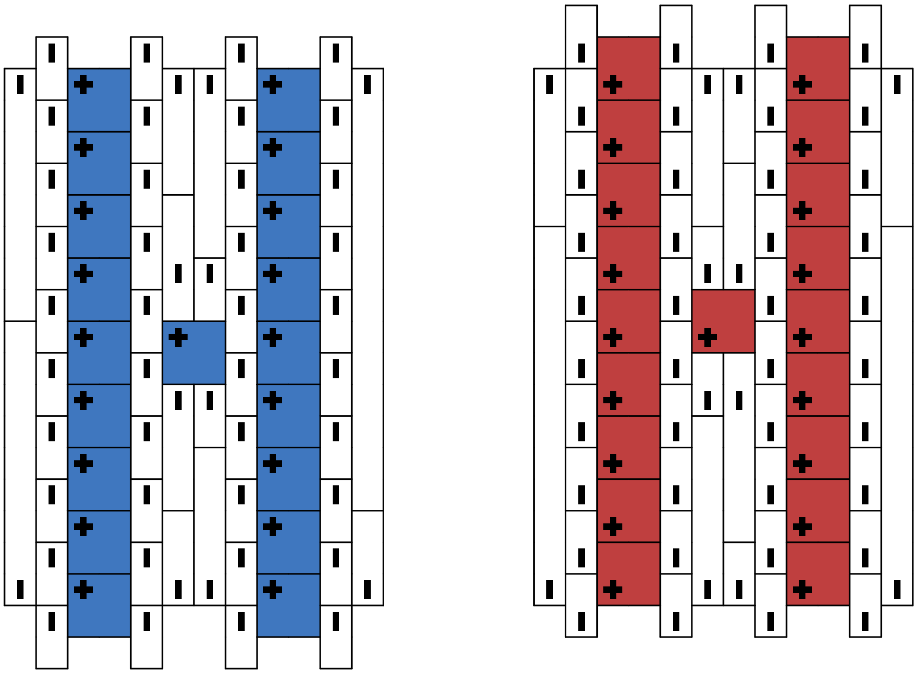


Figure 7 The variable gadget's profile table. Left: variable set to true. Right: variable set to false.

3.4 Clause Gadgets

The clause gadget, shown in Figure 8, interfaces with three wire gadgets representing the three literals of this clause. In the upper-left of the variable gadget is an internal wire, which we call the ***clause verification wire***. The only way to cover the top two cells of that wire is using the wire's top ⊞ clue. This is only possible when at least one of the wires is true, allowing a ***variable enforcement line*** (drawn in figures as a purple horizontal bar) to provide a parity shift to the clause verification wire. Otherwise, either those top two cells cannot be covered, or some other cell in the clause will not be covered, or there will be a four-corner violation.

The mandatory areas of the clause include all clues and cells shown in Figure 11 and optional areas consisting of the row of cells at the bottom of the gadget, specifically the set of cells under the ◫ clue lines at the bottom of the gadget.

Each of the three wires in this gadget has two intended solutions: true or false. In Figure 11, the wire is blue if it represents true and red if it represents false. The leftmost wire behaves somewhat differently from the others because it is closest to the clause verification wire.

Importantly, the clause gadget can be expanded horizontally such that the variable wires can be spaced an arbitrary amount beyond the width of the base gadget shown in Figure 11. The columns between the literal wires in the clause gadget can be expanded an arbitrary number of columns. Such an example expansion is shown in Figure 9. In this example, the columns have been expanded such that the entire gadget is wider by 4 columns and the number of columns between each literal in the gadget has been expanded by 2 columns.

▶ **Lemma 3.10.** *If any wire is in the* false *configuration, then the variable enforcement line corresponding to the wire will not be able to go across the gadget.*

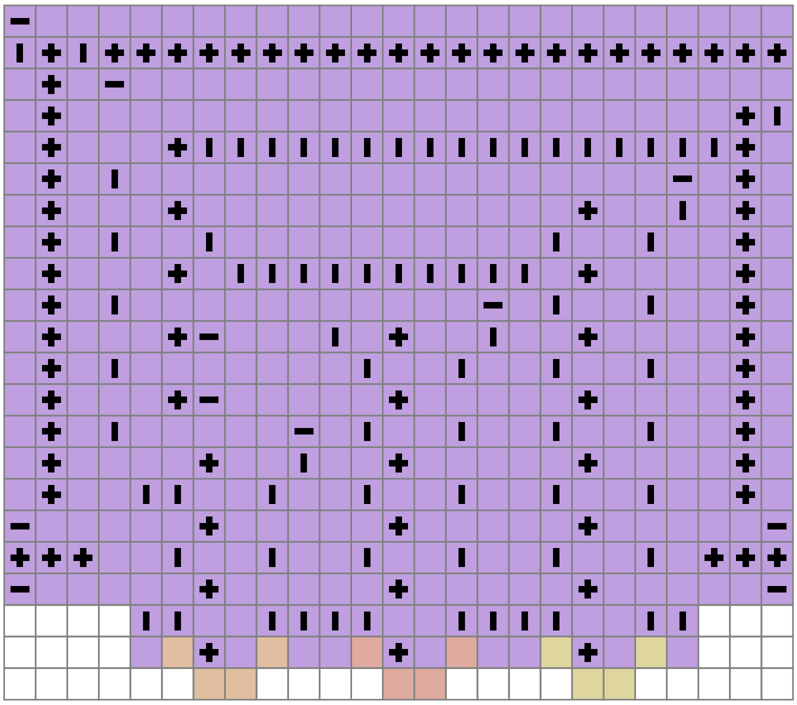


Figure 8 An unsolved clause gadget. Mandatory area is purple and optional areas are brown.

Proof. If a wire is in the false configuration, then there exists at two cells on the top of the wire that need to be covered. These two cells can be covered in two different ways. We first prove this lemma for the leftmost wire and then prove the lemma for the other wires since the leftmost wire is different from the others. In this case, the only way to cover the two cells is with a 2×2 square (see Figure 10), blocking the variable enforcement line from crossing the top of the wire.

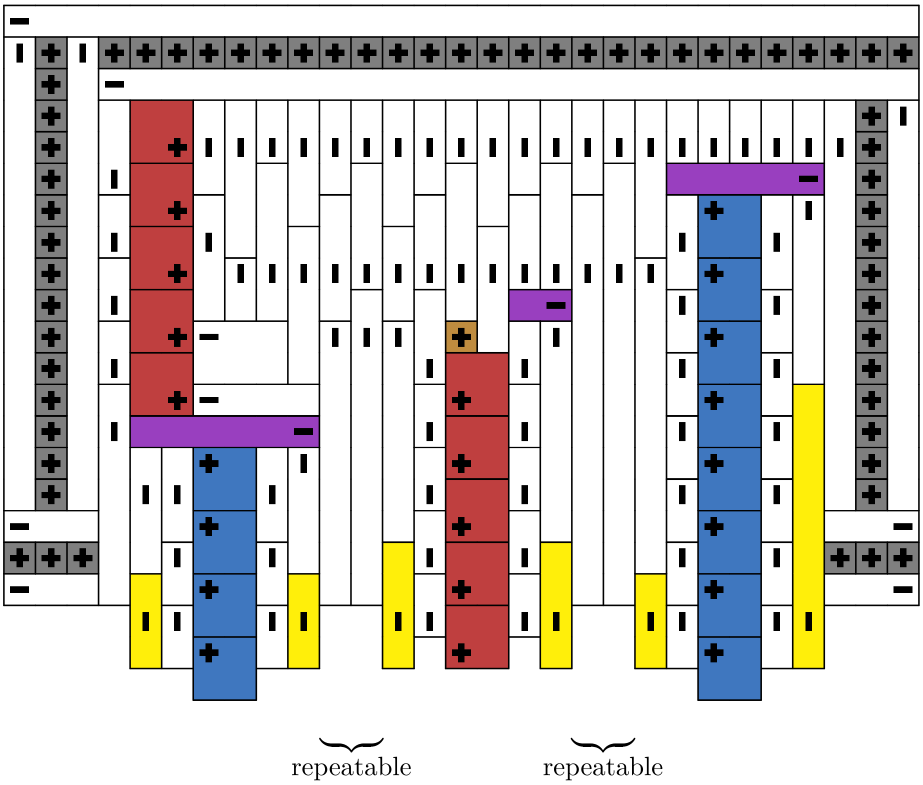
For the other two wires, the top two cells can be covered in only two ways. Either a 1×1 square covers one of the two cells and a vertical line from the top covers the other cell or vice versa (see Figure 11).

No other configurations are available that does not violate the four-corner constraint. Thus, this configuration prevents the corresponding variable enforcement lines from going across the gadget. ◀

▶ **Corollary 3.11.** *When all wires in the gadget are false, the puzzle does not have a solution.*

Proof. By Lemma 3.10, no variable enforcement line can go across the gadget if all wires are false. In order to solve the puzzle presented by the gadget, the top two cells of the clause verification line must be covered. These two cells cannot be covered by the horizontal line on top of them nor can they be covered by the vertical lines beside them. Thus, they must be covered by the 2×2 square formed in the clause verification line. However such a square will either leave a cell in the middle of the clause verification line uncovered or will leave the bottom two cells of the line uncovered. In this case, no configurations exist in covering these bottom two cells without violating the four-corner rule. See Figure 11a. Thus, the gadget is unsatisfiable if all wires into the gadget are false. ◀

▶ **Lemma 3.12.** *If at least one of the wires entering the clause gadget is in the true configuration, then the clause gadget is locally solvable.*



🟨 **Figure 9** Example where the columns in between literal wires in the clause gadget have been expanded. The columns which are able to be repeated an arbitrarily number of times have been labeled as "repeatable" in the figure since they can be repeated an arbitrarily number of times to make the clause an arbitrary width.

Proof. In any wire is in the true configuration, then the variable enforcement line corresponding to the gadget will be able to go across the gadget. For the leftmost wire, the clause verification line will be in the configuration that ensures that all cells that need to be covered by the line are covered. Otherwise, the variable enforcement line will be able to cause the clause verification line to cover all the necessary cells. See Figures 11b to 11h. ◀

Using the above lemmas, we are able to prove the following properties of the profile table of the clause gadget.

▶ **Corollary 3.13.** *The profile table of the clause gadget is proper.*

▶ **Lemma 3.14.** *The profile table of the clause gadget is complete.*

Proof. The clause gadget's profile table contains all profiles shown in Figure 11 except for the all-false configuration shown in Figure 11a. By Corollary 3.11, the all-false configuration is not locally solvable. It remains to show the all-false configuration is locally impossible.

To do this, we show that no solution to a clause outside of this profile is able to solve any part of the all-false clause profile–essentially that the clause gadget is fully isolated from the rest of the puzzle. By design, no clue above, to the left of, or to the right of the clause can cover any of the cells that are left uncovered by the literals, because the row and columns of single-cell squares blocks any rectangles from reaching the uncovered cells.

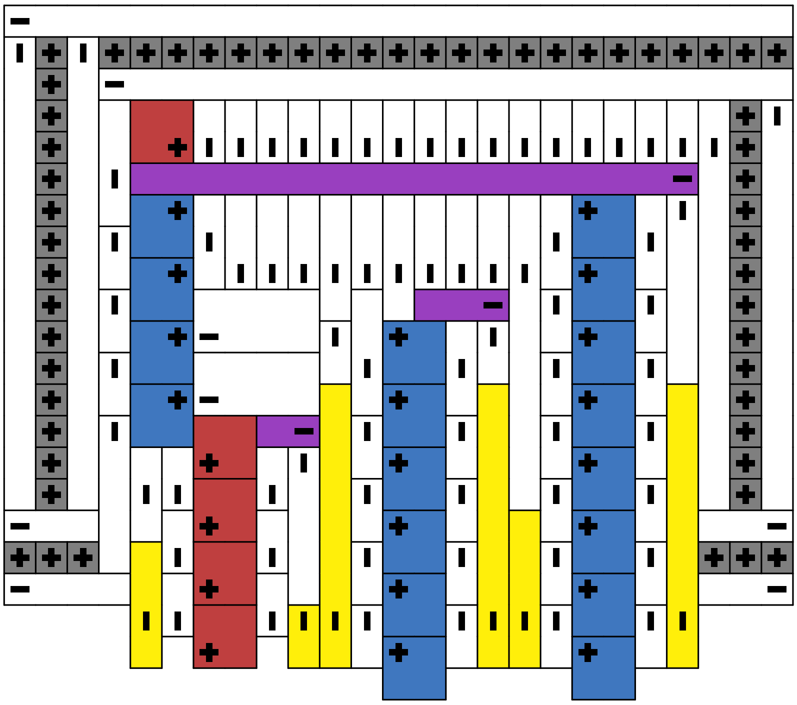


Figure 10 When the leftmost wire is set in the false configuration, the only way to cover the top two cells of the wire is with a 2 × 2 square that blocks the variable enforcement line.

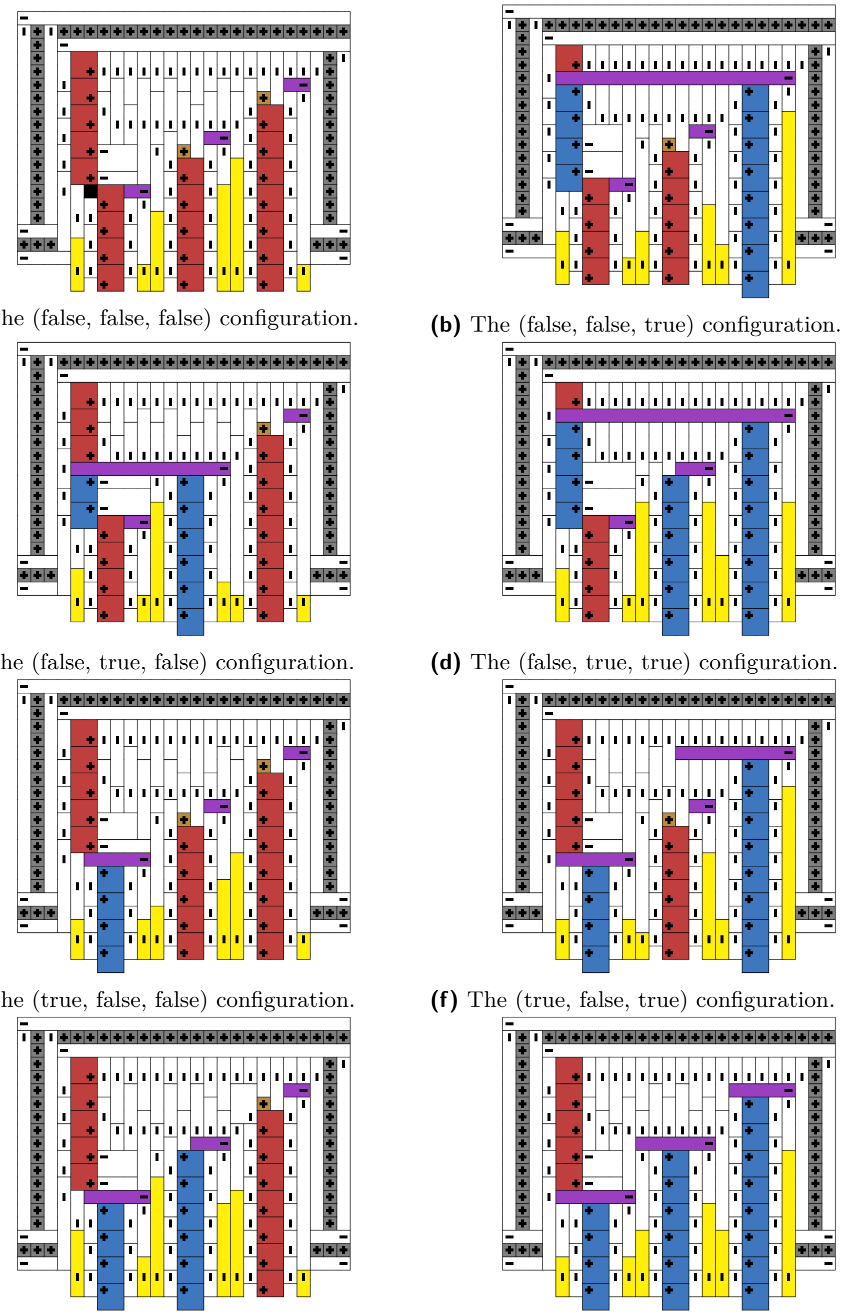
We now prove that no clues from the bottom of the gadget can help cover any of these cells. Such clues can only potentially cover the optional areas at the bottom of the gadget. We show that such clues cannot cover parts of the literal gadgets. By Lemma 3.7, there are only two possible configurations of the variable gadgets; thus, no other outside fillers can cover any cells in the incoming wires. Hence, no clues adjacent to the bottom of the gadget can help cover any part of the incoming wires.

Thus the all-false profile is locally impossible, so the profile table is complete. ◀

3.5 Layout, Sheathing, and Filler

In order to build the full Tatamibari instance corresponding to a planar rectilinear monotone 3SAT instance, we lay out the gadgets as shown in Figure 2: variable gadgets are positioned on a central line, while positive and negative clauses are positioned above and below respectively at heights corresponding with how many layers of clauses are nested below them, with wires running vertically from variables to clauses (both variable and clause gadgets can be extended arbitrarily far horizontally). Variable and clause gadgets have rectangular profiles (except for where the wires "plug in" to them). Variables and clauses have a uniform height, and for any two variable or clause gadgets, they are placed on exactly the same set of rows or they share no rows.

All wire gadgets in the puzzle produced by our reduction are safely placed; that is, no rectangle from a ◫ clue can reach the cells to the right of the top and bottom ⊞ clues in the wire. The only ◫ clues in those columns are in clause gadgets. The row of single-cell squares at the top of the clause gadget blocks any rectangles from extending upwards out of the clause gadget. If a rectangle from a ◫ clue in those columns of the clause gadget extends downward past the first ⊞ clue in the column to its left, the cell below that ⊞ clue cannot



(a) The (false, false, false) configuration.

(b) The (false, false, true) configuration.

(c) The (false, true, false) configuration.

(d) The (false, true, true) configuration.

(e) The (true, false, false) configuration.

(f) The (true, false, true) configuration.

(g) The (true, true, false) configuration.

(h) The (true, true, true) configuration.

Figure 11 The clause gadget. All configurations shown here except the all-false configuration in Figure 11a are in the clause gadget profile table. Clues highlighted in yellow also function as the "outer sheathing" protecting the wires closest to them (see Section 3.5). For the false wires, the only configuration that guarantees the two cells at the top are covered are the cases where one 1 × 1 square covers one (shown in brown) and a long rectangle extending from the top covers the other.

be covered by any clue, so rectangles cannot extend downward out of the clause gadget in those columns. Thus | clues from clause gadgets cannot interact with wire gadgets, so the wires are safely placed.

Because we want the solvability of the Tatamibari instance to depend only on solving the gadgets, we need to add ***filler*** clues that are always able to cover the areas outside the gadgets.

First, we set aside any cells horizontally adjacent to a wire gadget. These cells will be covered by the outer sheathing clues described in described in Section 3.2 and Section 3.3 and highlighted in yellow in Figure 9 and Figure 11. In the global solution, the areas assigned to the outer sheathing clues thus extend vertically outside their gadget. For the purposes of the filler algorithm, we consider the cells covered by the outer sheathing to be part of the wire gadget.

Each filler clue corresponds to a rectangular area of space between gadgets, formed by breaking each row into maximal horizontally contiguous strips between (and bordered by) the gadgets, then joining vertically contiguous strips into a single rectangle if they have the same width. The filler algorithm places a single clue in each of these rectangles (⎅, ⊟, or ⊞ depending on the rectangle's aspect ratio), placed arbitrarily inside (say, in the upper-right corner). See Figure 2 for an example. While it may be possible for the solver to use these clues differently than shown here, we only need to prove that if the solver does assign each rectangular area to its associated clue, it will cover the area.

The only potential problem lies in the possibility of a four-corner violation involving a filler rectangle. This can only happen where either (i) a corner of a filler rectangle meets a gadget and a wire coming from that gadget, or (ii) where two corners of filler rectangles meet along the edge of a gadget. If a corner of a filler section meets an edge of another filler section or the edge of the board there cannot be a four-corner violation.

Remark. There is a potential third problem case, where two wires are directly adjacent with only the outer sheathing (2 columns) between them (see Figure 8, which has this property). This can be dealt with in either of two ways: ensuring that no wires are directly adjacent to each other by stretching the instance horizontally, or noting that the meeting points of the outer sheathing of the two adjacent wires can be adjusted to not produce a four-corner violation between them.

▶ **Proposition 3.15.** *If the gadgets can all be satisfied, the filler clues can also be satisfied.*

Proof. Each filler clue will be satisfied by a rectangle covering its entire associated area; the cells horizontally adjacent to wires will be filled by two width-1 vertical rectangles from the outer sheathing clues, one coming from the clause gadget above and the other coming from the variable gadget below. The meeting point between the two outer sheathing rectangles can be adjusted as needed to avoid a four-corner violation. As mentioned, we have only two problem cases: (i) a corner of the filler rectangle meets a gadget and protruding wire; and (ii) corners of two sections meet on the side of a wire. Because both cases involve the side of a wire, we can avoid violations in either case by appropriately adjusting the meeting point of the sheathing clues.

(i) To avoid having a corner where the corner of the filler section meets the wire and gadget, the meeting point of the two sheathing clues can be placed on the edge (not corner) of the filler section, thus avoiding a four-corner violation since the corner of the filler section meets the edge of one of the sheathing rectangles.

(ii) As long as the meeting point of the two sheathing rectangles of the wire is not at the point where the two filler sections meet, there is no four-corner violation. The meeting point can trivially be placed on the side of a filler section (while still respecting the parity of the wire as expressed by the inner sheathing).

Therefore, since the sheathing can always be adjusted to accommodate filler rectangles, the satisfiability of the Tatamibari instance depends only on the gadgets. ◀

3.6 Finale

Now we can show that Tatamibari is NP-hard. Let f be the reduction, which takes an instance Φ of planar rectilinear monotone 3SAT and returns a Tatamibari instance $f(\Phi)$; we want to show:

▶ **Proposition 3.16.** *If Φ has n variables and m clauses, then $f(\Phi)$ has size polynomial in $n+m$, and can be computed in time polynomial in $n+m$.*

Proof. Our construction expands the given planar rectilinear monotone 3SAT instance by a constant factor. Therefore it suffices to prove that planar rectilinear monotone 3SAT is strongly NP-hard when given the coordinates of the rectilinear drawing. Indeed, the height of the drawing is $O(m)$ and the width of the drawing is $O(e)$ if the graph has e edges, which is $O(m+n)$ by planarity. ◀

▶ **Proposition 3.17.** *If Φ has a solution, then $f(\Phi)$ also has a solution.*

Proof. We begin by taking the solution to Φ and setting the variable gadgets' profiles according to those values; by Lemma 3.9, they will all be locally solvable. By Lemma 3.12, since each clause gadget is connected to wires representing variables which satisfy the clause, there must be a solution to the clause gadget. Furthermore, by Proposition 3.15, if the gadgets are satisfiable then the rest of the space can be filled without contradiction, producing a solution to $f(\Phi)$. ◀

▶ **Proposition 3.18.** *If Φ has no solution, then $f(\Phi)$ also has no solution.*

Proof. We prove the equivalent statement that if $f(\Phi)$ has a solution, then Φ must also have a solution.

First, we prove that any solution to $f(\Phi)$ must correspond to some setting of the variables $x_1, \ldots, x_n$ of Φ. This is a consequence of Lemma 3.8, which shows that all wires in a single variable gadget must carry the same value, which is then taken as the setting for that variable.

Next, we have to show that these settings of the variables x_i are a solution of Φ; to do this, note that by Corollary 3.2 each wire ending in a clause gadget must carry its value into this clause gadget; and by Corollary 3.11 and Lemma 3.12 there is a solution to the clause gadget if and only if the wires represent values which satisfy the clause.

Thus, the values of the variable gadgets must be a solution to Φ. ◀

The above three propositions imply our desired result:

▶ **Theorem 3.19.** *Tatamibari is (strongly) NP-hard.*

Because a given Tatamibari solution can be trivially checked in polynomial time, this theorem implies that Tatamibari is NP-complete.

4 Font

Figure 12 shows a series of twenty-six 10 × 10 Tatamibari puzzles that we designed, whose unique solutions shown in Figure 13 reveal each letter A–Z. To represent a bitmap image in the solution of a Tatamibari puzzle, we introduce two colors for clues, light and dark, and similarly shade the regions corresponding to each clue. As shown in Figure 13, the letter is

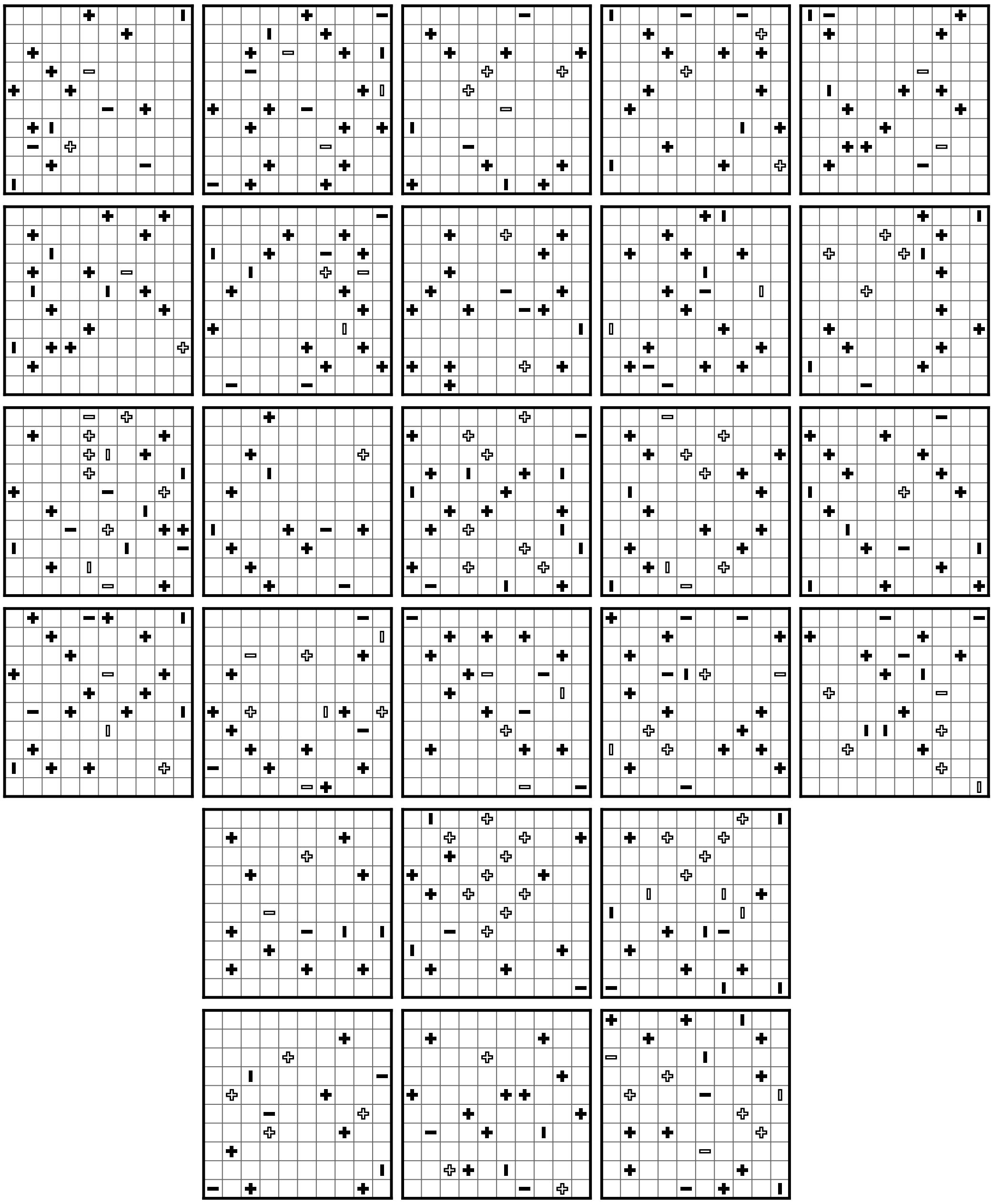


Figure 12 Puzzle font: each puzzle has a unique solution whose regions for dark clues (shown in Figure 13) form the shape of a letter.

drawn by the dark regions from dark clues. These puzzles were designed by hand, using our SAT-based solver [4] to iterate until we obtained unique solutions. The font is also available online.[2]

[2] http://erikdemaine.org/fonts/tatamibari/

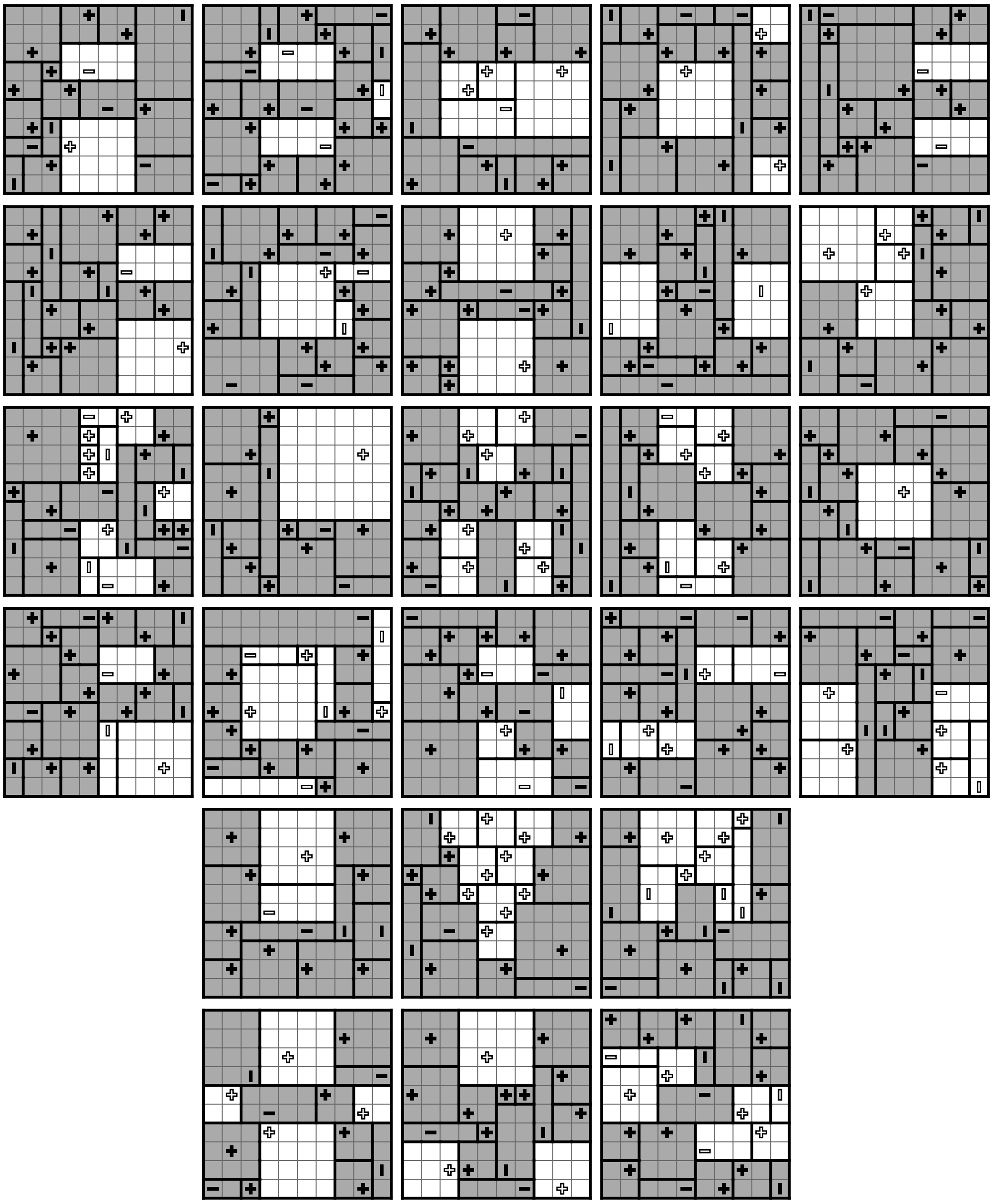


Figure 13 Solved font: unique solutions to the puzzles in Figure 12.

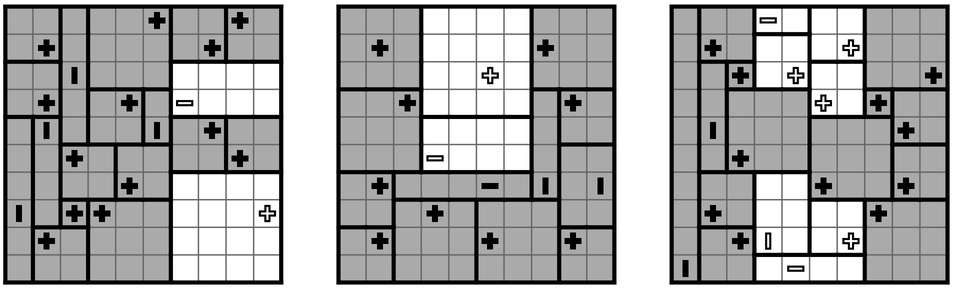


Figure 14 Solution to Figure 1.

5 Open Problems

There remain interesting open questions regarding the computational complexity of Tatamibari. When designing puzzles, it is often desired to have a single unique solution. We suspect that Tatamibari is ASP-hard (NP-hard to determine whether it has another solution, given a solution), and that counting the number of solutions is #P-hard. However, our reduction is far from parsimonious. Some rework of the gadgets, and a unique filler between gadgets, would be required to preserve the number of solutions.

We could ask about restrictions or natural variations of Tatamibari. For example, we are curious whether a Tatamibari puzzle with only ⊞ clues, or only ◫ clues, remains hard. We have also wondered about the version of the puzzle without the four-corner constraint. Although initially we thought of the four-corner constraint as a nuisance to be overcome in our reduction, our final proof uses it extensively and centrally.

References

1 Zachary Abel, Jeffrey Bosboom, Erik D. Demaine, Linus Hamilton, Adam Hesterberg, Justin Kopinsky, Jayson Lynch, and Mikhail Rudoy. Who witnesses The Witness? Finding witnesses in The Witness is hard and sometimes impossible. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 3:1–3:21, La Maddalena, Italy, June 2018.

2 Aaron B. Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Zig-zag numberlink is NP-complete. *Journal of Information Processing*, 23(3):239–245, 2015. doi:10.2197/ipsjjip.23.239.

3 Aviv Adler, Michael Biro, Erik Demaine, Mikhail Rudoy, and Christiane Schmidt. Computational complexity of numberless Shakashaka. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG 2015)*, Kingston, Canada, August 2015.

4 Aviv Adler, Jeffrey Bosboom, Erik D. Demaine, Martin L. Demaine, Quanquan C. Liu, and Jayson Lynch. Z3-based Tatamibari solver, and figures from Tatamibari NP-hardness paper, 2020. Includes inputs for the gadgets in this paper. URL: https://github.com/jbosboom/tatamibari-solver.

5 Addison Allen, Daniel Packer, Sophia White, and Aaron Williams. Pencils and Sto-Stone are NP-complete [paper in review], 13 March 2018. URL: https://www.researchgate.net/project/Computational-Complexity-of-Video-Games-and-Puzzles/update/5aa7c402b53d2f0bba57bfb8.

6 Walker Anderson, Erik D. Demaine, and Martin L. Demaine. Spiral galaxies font. In Jennifer Beineke and Jason Rosenhouse, editors, *The Mathematics of Various Entertaining Subjects (MOVES 2017)*, volume 3, pages 24–30. Princeton University Press, 2019.

7 Daniel Andersson. HIROIMONO is NP-complete. In *Proceedings of the 4th International Conference on FUN with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 30–39, 2007. URL: http://www.springerlink.com/content/h31jq82185n0618h/?p=408092624f724be298d11ec22a3da382&pi=4.

8 Daniel Andersson. Hashiwokakero is NP-complete. *Information Processing Letters*, 109(19):1145–1146, 2009.

9 Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In My T. Thai and Sartaj Sahni, editors, *Proceedings of the 16th Annual International Conference on Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 216–225, July 2010.

10 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Budapest, Hungary, 2008. doi:10.1007/978-3-540-78800-3_24.

11 Erik D. Demaine and Martin L. Demaine. Fun with fonts: Algorithmic typography. *Theoretical Computer Science*, 586:111–119, June 2015.

12 Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno. Computational complexity and an integer programming model of Shakashaka. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E97-A(6):1213–1219, 2014. URL: `http://hdl.handle.net/10119/12147`.

13 Erich Friedman. Corral puzzles are NP-complete. `http://www.stetson.edu/~efriedma/papers/corral/corral.html`, August 2002.

14 Erich Friedman. Pearl puzzles are NP-complete, August 2002. URL: `http://www.stetson.edu/~efriedma/papers/pearl/pearl.html`.

15 Erich Friedman. Spiral Galaxies puzzles are NP-complete, March 2002. URL: `https://www2.stetson.edu/~efriedma/papers/spiral/spiral.html`.

16 Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, July 1977. `doi:10.1145/1008354.1008356`.

17 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation.* A K Peters, July 2009.

18 Markus Holzer, Andreas Klein, and Martin Kutrib. On the NP-completeness of the NURIKABE pencil puzzle and variants thereof. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, pages 77–89, Isola d'Elba, Italy, May 2004.

19 Markus Holzer and Oliver Ruepp. The troubles of interior design—a complexity analysis of the game Heyawake. In *Proceedings of the 4th International Conference on FUN with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 198–212, 2007. URL: `http://www.springerlink.com/content/77t44731124j6427/?p=421c0ae5f9364a0880436edd93980bd7&pi=17`.

20 Ayaka Ishibashi, Yuichi Sato, and Shigeki Iwata. NP-completeness of two pencil puzzles: Yajilin and Country Road. *Utilitas Mathematica*, 88:237–246, 2012.

21 Chuzo Iwamoto. Yosenabe is NP-complete. *Journal of Information Processing*, 22(1):40–43, 2014. `doi:10.2197/ipsjjip.22.40`.

22 Jonas Kölker. Kurodoko is NP-complete. *Journal of Information Processing*, 20(3):694–706, 2012. `doi:10.2197/ipsjjip.20.694`.

23 Kouichi Kotsuma and Yasuhiko Takenaga. NP-completeness and enumeration of Number Link puzzle. *IEICE Technical Report*, 109(465):1–7, March 2010. URL: `http://ci.nii.ac.jp/naid/110008000705/en/`.

24 Brandon McPhail. The complexity of puzzles. Undergraduate thesis, Reed College, Portland, Oregon, 2003. URL: `http://www.cs.umass.edu/~mcphailb/papers/2003thesis.pdf`.

25 Brandon McPhail. Light Up is NP-complete. `http://www.mountainvistasoft.com/docs/lightup-is-np-complete.pdf`, 2005.

26 Brandon McPhail. Metapuzzles: Reducing SAT to your favorite puzzle. CS Theory talk, December 2007.

27 Nikoli Co., Ltd. タタミバリ（仮題） (Tatamibari (temporary title)). *Puzzle Communication Nikoli*, 107, 10 June 2004. See `https://www.nikoli.co.jp/ja/publication/various/nikoli/back_number/nikoli107/` for a table of contents.

28 Nikoli Co., Ltd. パズル通信ニコリ > オモパリスト (Puzzle Communication Nikoli > Omopa List), 2020.

29 Nikoli Co., Ltd. Nikoli puzzles, 2020. URL: `http://www.nikoli.co.jp/en/puzzles/`.

30 Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, May 1996. URL: `http://www.phil.uu.nl/~oostrom/oudonderwijs/cki20/02-03/japansepuzzles/complexity.ps`.

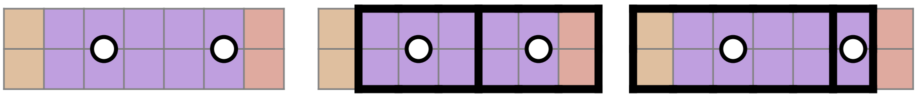
31 Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, University of Tokyo, Tokyo, Japan, January 2003. URL: `http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.pdf`.

32 Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, E86-A(5):1052–1060, 2003. Also IPSJ SIG Notes 2002-AL-87-2, 2002. URL: `http://ci.nii.ac.jp/naid/110003221178/en/`.

A Example: Spiral Galaxies

As an example of the gadget area hardness framework, we show how the NP-hardness proof for Spiral Galaxies from [15] can be described using the framework. A Spiral Galaxies puzzle is a rectangular grid with clues in some grid cells or on some grid lines. The goal is to partition the puzzle into areas with a single clue per area such that the area is rotationally symmetric about the clue.

We reduce from planar[3] Boolean circuit satisfiability. We have a wire gadget, a variable gadget, NOT and AND gadgets, a fanout (wire duplicator) gadget, and a vertical shift gadget. We lay out these gadgets to overlap in their optional areas (only), and communicate a truth value in whether the optional area is covered or not.



(a) Unsolved wire gadget **(b)** Wire carrying true signal: 3 × 2 rectangles **(c)** Wire carrying false signal: alternating 1 × 2 and 5 × 2 rectangles

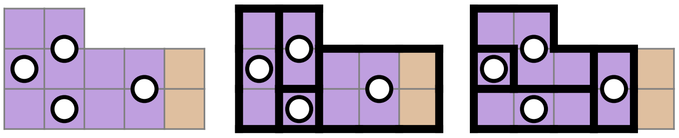
Figure 15 Spiral Galaxies wire and its profile table (true and false solutions).

Wire. The wire gadget consists of repeating pairs of clues three grid units apart. There are two gadget solutions, shown in Figure 15: repeating 3 × 2 rectangles, in which case the wire covers the right optional area, and alternating 1 × 2 and 5 × 2 rectangles, in which case the wire covers the left optional area. The wire carries a true signal when it covers the right optional area and false when it covers the left optional area. The wire gadget can be extended to arbitrary length in units of two clues. (The proof in [15] does not explicitly state this parity requirement, but the gate gadgets assume the true signal protrudes into the gadget to cover the optional input area and the false signal does not.)

Boolean circuit satisfiability requires the circuit produce a true output. We can force a wire to be true simply by terminating it. Because the wire has height two, any filler clues to the right of the wire cannot cover area in the wire gadget, so the wire must end in a 3 × 2 rectangle to cover the right optional area, forcing the rest of the wire to also carry a true signal.

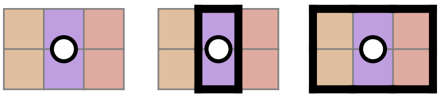
Variable. The variable gadget is shown in Figure 16. There are two gadget solutions, one leaving the optional area uncovered (so the adjacent wire is set to true) and the other covering it (so the adjacent wire is set to false). Choosing one solution or the other corresponds to assigning true or false to the variable.

[3] Friedman's proof [15] provides a crossover gadget, but it is not necessary because AND and NOT build a crossover [16].



(a) Unsolved variable gadget **(b)** Variable set to true **(c)** Variable set to false

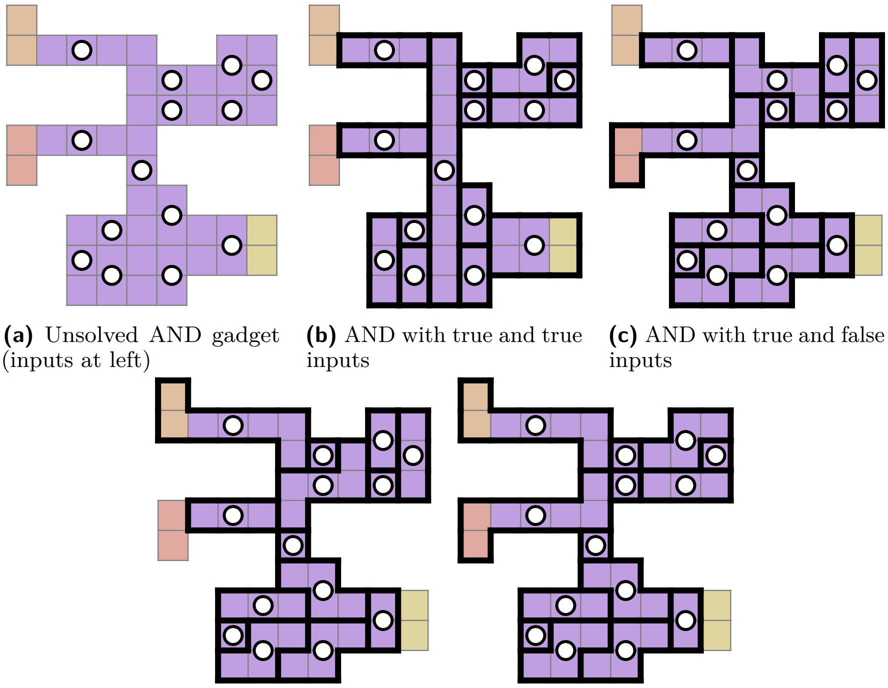
Figure 16 Spiral Galaxies variable and its profile table (true and false solutions).



(a) Unsolved NOT gadget **(b)** NOT gadget converting true to false **(c)** NOT gadget converting false to true

Figure 17 Spiral Galaxies NOT gadget and its profile table.

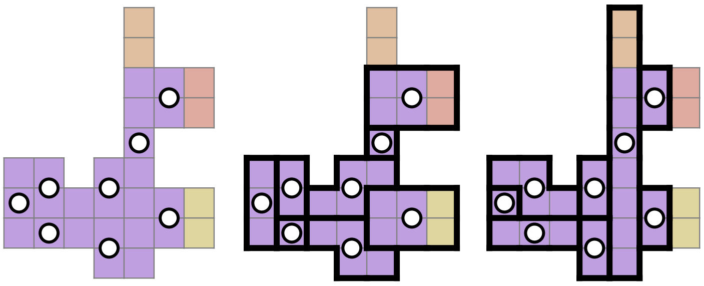
NOT. The NOT gadget is shown in Figure 17. If the left optional area is covered by the input wire (carrying a true signal), the clue in the NOT gadget must cover a 1 × 2 rectangle, so the right optional area must be covered by the output wire carrying a false signal. If the left optional area is uncovered (when the input wire is false), the clue in the NOT gadget covers both optional areas, so the output wire must carry a true signal. Thus the NOT gadget inverts the wire's signal.



(a) Unsolved AND gadget (inputs at left) **(b)** AND with true and true inputs **(c)** AND with true and false inputs

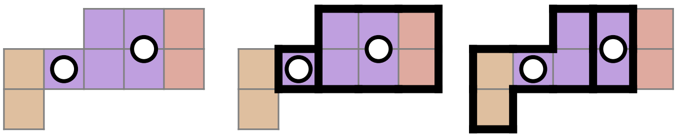
(d) AND with false and true inputs **(e)** AND with false and false inputs

Figure 18 Spiral Galaxies AND gadget and its profile table.



(a) Unsolved fanout gadget (input at top) **(b)** Fanout gadget duplicating true wire **(c)** Fanout gadget duplicating false wire

Figure 19 Spiral Galaxies fanout gadget and its profile table.



(a) Unsolved shift gadget **(b)** Shift gadget shifting a true wire **(c)** Shift gadget shifting a false wire

Figure 20 Spiral Galaxies upward shift gadget and its profile table; the downward shift gadget is this gadget flipped vertically.

AND. The AND gadget is shown in Figure 18. When both inputs are true, both of the left optional areas are covered by the wire, so the clues to the right of the optional area are rectangles and the clue at the center of the gadget is a long vertical rectangle, allowing the right optional area to be covered, propagating a true signal from the gadget. When either or both of the inputs are false, one or both of the left optional areas must be covered by the clue(s) to the right of the areas, blocking the central clue from covering a vertical rectangle, preventing the right optional area from being covered, thus propagating a false signal from the gadget.

Fanout. Like the AND gadget, the fanout gadget (Figure 19) is also based around forming or not forming a vertical rectangle. The upper optional area is the input. When it is covered by the input wire (a true signal), the central clue cannot form a vertical rectangle, so the upper-right optional area must be covered by the clue to its left, and because the bottom cell in the central column is covered by the clue to its upper-left, the lower-right optional area must also be covered by the clue to its left. When the upper optional area is not covered by the input wire, it must be covered by the central clue forming a vertical rectangle, so the output optional areas cannot be covered by the clues to their left.

Shift. Because variable and gate outputs are on the right and gate inputs are on the left, we do not need a turn gadget, but we do need to shift wires vertically, which is done using the shift gadget. An upward shift gadget is shown in Figure 20; the downward shift gadget is that gadget's reflection across the horizontal axis. When the input wire is true, the input wire covers the left optional area, so the left clue is covered by a single cell and the right clue covers the right optional area, propagating true on the output. When the input wire is false, the left clue covers the left optional area and forces the right clue to be a 1 × 2 rectangle, leaving the right optional area uncovered, propagating false on the output.

Layout. Friedman's proof in [15] omits discussion of layout, but we sketch a layout algorithm here. We start with a grid embedding of the input planar Boolean circuit. We scale the grid by at least 6 so that our wire gadget fits for unit-length wires, but possibly by a greater factor if the grid embedding has long vertical segments, because our shift gadget consumes horizontal distance to move vertically.

Filling algorithm. The filling algorithm places a clue in the center of every cell that isn't part of a gadget, forcing them to be covered by single-cell areas. Filler clues could only cover area in a gadget if two cells in the gadget area are separated by one filler clue and those cells do not themselves have clues. This is avoided in all gadgets by ensuring all gadget cells that are separated by filler are separated by two or more filler cells, so only local gadget solutions are possible.

Composition algorithm. The local gadget solutions are already consistent with each other, so to form an area assignment for the entire puzzle, the composition algorithm simply takes the local gadget solutions and assigns each filler clue to the cell containing it.

Proper and complete profile tables. The profile tables are proper because they contain only proper profiles. Because the filler clues cannot cover area in the gadgets, we can verify by case analysis that the profile tables are complete (all other profiles are locally impossible). This completes the proof.

Collaborative Procrastination

Aris Anagnostopoulos
Sapienza University of Rome, Italy
http://aris.me
aris@diag.uniroma1.it

Aristides Gionis
KTH Royal Institute of Technology, Stockholm, Sweden
https://www.kth.se/profile/argioni
argioni@kth.se

Nikos Parotsidis
University of Copenhagen, Denmark
https://sites.google.com/view/nikosparotsidis
nipa@di.ku.dk

Abstract

The problem of inconsistent planning in decision making, which leads to undesirable effects such as procrastination, has been studied in the behavioral-economics literature, and more recently in the context of computational behavioral models. Individuals, however, do not function in isolation, and successful projects most often rely on team work. Team performance does not depend only on the skills of the individual team members, but also on other collective factors, such as team spirit and cohesion. It is not an uncommon situation (for instance, experienced by the authors while working on this paper) that a hard-working individual has the capacity to give a good example to her team-mates and motivate them to work harder.

In this paper we adopt the model of Kleinberg and Oren (EC'14) on time-inconsistent planning, and extend it to account for the influence of procrastination within the members of a team. Our first contribution is to model collaborative work so that the relative progress of the team members, with respect to their respective subtasks, motivates (or discourages) them to work harder. We compare the total cost of completing a team project when the team members communicate with each other about their progress, with the corresponding cost when they work in isolation. Our main result is a tight bound on the ratio of these two costs, under mild assumptions. We also show that communication can either increase or decrease the total cost.

We also consider the problem of assigning subtasks to team members, with the objective of minimizing the negative effects of collaborative procrastination. We show that whereas a simple problem of forming teams of two members can be solved in polynomial time, the problem of assigning n tasks to n agents is **NP**-hard.

2012 ACM Subject Classification Applied computing → Economics; Applied computing → Sociology

Keywords and phrases time-inconsistent planning, computational behavioral science, collaborative work, collaborative environments

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.2

Funding *Aris Anagnostopoulos*: Supported by the ERC Advanced Grant 788893 AMDROMA, the EC H2020RIA project "SoBigData++" (871042), and the MIUR PRIN project ALGADIMAR "Algorithms, Games, and Digital Markets".
Aristides Gionis: Supported by three Academy of Finland projects (286211, 313927, 317085), the EC H2020RIA project "SoBigData++" (871042), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.
Nikos Parotsidis: Supported by Grant Number 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 2; pp. 2:1–2:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

> Procrastination has taught me how to do 30 minutes of work in 8 hours and 8 hours of work in 30 minutes.
>
> – anonymous internet user

The synthesis of teams is a fundamental activity within organizations. The importance of teams in the production of knowledge is increasing. For instance, in the context of scientific research, teams typically produce more frequently cited research than single individuals [15]. Furthermore, it has been observed that simply putting together the best individuals does not necessarily create a great team [11], as there are aspects characterizing effective group members and successful collaborations that are not evident in an individual's performance. When forming teams, it is necessary to take many aspects of the team into account, such as diversity, learning, and cohesion. Although all the aforementioned characteristics play an important role in the performance of a team, they fail to characterize how team dynamics evolve when individuals tend to procrastinate.

In many cases, a project is divided in subtasks, which are assigned to the members of a team. Such a division facilitates cooperation and takes advantage of the different skillsets of the team members. In such situations, typically the final outcome of the project depends on the completion of the subtasks that are assigned to the team members. In this work we assume that the individual members of the team work independently on the subtasks that have been assigned to them, they are aware of the progress that has been made by their teammates, and they do not help each other by working on others' tasks, lacking either the expertise or the incentive to do so. This is a reasonable assumption, especially in the case that each subtask requires different skillsets.

To motivate our setting, consider the following example.

Example. A software company gets assigned a project and the project manager gathers a team of engineers to form a team and work on the project. The project has a number of different subtasks and the project manager recruits one person with the required skills for each subtask (e.g., back-end development, data analytics, user interface). Success in the project depends on completing all subtasks; if one subtask fails, the whole project fails. The team holds regular meetings, sets milestones, discusses problems that occur, and reports progress made in the different subtasks. The collective progress affects the motivation and performance of the team members. An engineer who would normally be motivated to work and would rarely procrastinate might feel unmotivated if the others do not make progress on their subtasks. Conversely, if everyone makes good progress, an engineer who is prone to procrastination might fear that the project will fail because of him and he would put his best effort to keep up with the team. □

In this example it is clear that progress by motivated individuals may help to motivate others. At the same time, motivated individuals can be discouraged to make further progress if they realize that their reward will be unfairly proportional to their effort. Similarly for procrastinating individuals, their "free-ride" attitude may discourage other team members, or they can get motivated by realizing that they are the ones who keep the team behind. Therefore, the overall process is governed by complex dynamics.

Motivated by this discussion, the questions that we study in this paper are the following:

Q1: How can we model interactions among members of a team who work on the same project, such as to capture the dynamics for motivating (or demotivating) each other?

Q2: What are the effects of such a model of interaction among team members, and to what extent the performance of the team can be sped up or slowed down?

Q3: Can we assign optimally team members to the subtasks of a project such as to take advantage of the interactions among the team members and to minimize the total cost of completing a project?

The time-inconsistent planning model. To model procrastination of individual team members we use the time-inconsistent planning model [1, 13, 14]. Here we adopt the formulation introduced by Kleinberg and Oren [8]. We refer to individual team members as agents. According to this model, the progress of an agent for a particular task is represented as a single-source–single-sink directed acyclic graph. The graph simulates a discrete-time process. Each vertex in the graph represents the current state in the project and the progress made so far. An agent being at vertex u at time t picks an edge (u, v) going out of u and moves to vertex v at time $t + 1$. The source vertex represents the start of the project, and the sink vertex the completion of the project. Edge weights model the effort required to move along the edges, and agents are assumed that they try to minimize their total effort to complete the project. An agent with no bias will move from start to completion by following a shortest path from source to sink. To simulate procrastination, the model assumes a present-time bias, where agents perceive the cost at present time higher than what it is in reality. In particular, at any given time, the weights of the outgoing edges from the current vertex are multiplied by a factor $b \geq 1$. The agent calculates the shortest path to the sink using the inflated weights for the next-step edges. This leads agents to bias their choice of next-step edges towards low-cost edges, and as a result they follow paths whose total cost is larger than the cost of the shortest path.

The proposed model. To model collaborative procrastination, and provide an answer to **Q1**, we extend the time-inconsistent planning model, to account for interaction among team members. In particular, we assume that the overall project is divided in subtasks, each team member is assigned to one of the subtasks, and each subtask is represented by a single-source–single-sink directed acyclic graph, which is used to model the actions and progress of the assigned agent to the subtask. We assume that each agent i has a present-time bias b_i. In addition to the original model, we assume that an agent i takes steps towards completing her subtask, the fraction $q_i(t) \in [0, 1]$ capturing the progress made up to time t. The fraction $q_i(t)$ is known to agent i, as well as to all other agents. Given two agents i and j, the difference $q_i(t) - q_j(t)$ expresses the difference in their progress in their respective subtasks, at time t. If $q_i(t) - q_j(t) > 0$ agent i is *ahead* in her subtask and she may feel discouraged by the fact that j has not worked as hard. Conversely, agent j is *behind* and he may feel motivated to catch up. To capture the dynamics of this interaction, we propose to introduce a multiplicative factor $\gamma_i^{q_i(t)-q_j(t)}$ in the present-time bias factor of i, for some $\gamma_i \geq 1$. The effect of our model is that, in addition to the personal present-time bias factor b_i, which captures the tendency of i for procrastination, agents further slow down if they have done more progress than their peers, or speed up if they have done less progress.

Our results. We define formally the *collaborative-procrastination* model, outlined above. To answer research question **Q2** we consider the total cost required by the team to complete a task when they interact and their behavior follows the collaborative procrastination model, and we compare this with the cost that would be required if each agent was working independently on their subtasks. We focus on grid graphs, where at each state of the subtask an agent has two options: make progress or procrastinate. This family of graphs reportedly captures the worst-case task graphs that exhibit the less efficient planning by an agent.

To avoid pathological cases, we consider that the subtask graphs satisfy certain natural assumptions, as proposed by Gravin et al. [7] . Namely, we consider a *bounded distance* property, where in all subtask graphs the optimal path to complete the subtask from any state is never worse than the optimal path from the initial state, and a *monotonicity* property, which ensures that in each subtask graph the cost to complete the subtask does not increase over time. We express our results with respect to the size (n) of the subtask graph, and the number of agents (k) in the team.

Our main result is to show that, assuming the bounded-distance and monotonicity properties, the total cost paid by all agents in the collaboration model, compared to the total cost paid by all agents when they work in isolation, cannot increase more than a factor of $\Theta(n)$. Furthermore, we provide an example, which indicates that this bound is tight.

It is also possible that collaboration helps the overall team performance. We show that, under an additional (mild) assumption for subtask graphs, namely, that procrastinating is less costly at the current step than taking an action towards completing the task, our collaborative model can lead to speeding up the time to complete the overall task by a factor of $\Theta(n)$.

Finally, we turn to our research question **Q3**, for assigning team members to subtasks such as to minimize the total cost of completing the project. We consider a simple version of the problem when the subtask graphs are fixed for all agents, and each agent is characterized by their own present-time bias parameter, and interaction parameters with other agents. We show that even this simple version of the problem is **NP**-hard. We leave as an open problem the design of an efficient approximation algorithm.

2 Model

Our model builds on the graph-theoretic planning model that was introduced by Kleinberg and Oren [8] for a single agent. According to that model, a task is represented by a directed acyclic graph $G = (V, E)$, where each vertex represents a possible state of the task at a specific time point. In this paper we work with a specific family of task graphs that have a grid structure. Specifically, we identify every vertex v_t^ℓ by its time step t and an index $\ell \in \{0, 1, \ldots, \ell_{\max}\}$ indicating the progress that has been made so far towards completion of the task. We assume that no agent is failing her task, even at the expense of heavy cost by the last-minute work. Hence, the vertex set of the task graph consists of all vertices v_t^ℓ with $\ell \geq t$ and $t_{max} - t \geq \ell_{max} - \ell$. See Figure 1 for two examples of graphs.

There is a distinguished start vertex $\sigma = v_0^0$ and a target vertex $\tau = v_{t_{\max}}^{\ell_{\max}}$ that represent the starting point and the completion of the task, respectively. The edge set E contains the following directed edges: (1) an edge $(v_t^\ell, v_{t+1}^{\ell+1})$ for each vertex v_t^ℓ such that for $\ell = 0, \ldots, \ell_{\max} - 1$, $t = \ell, \ldots, t_{\max} - 1$, (2) an edge (v_t^ℓ, v_{t+1}^ℓ) for each vertex v_t^ℓ such that $t_{max} - t \geq \ell_{max} - \ell$ (this condition ensures that the agent does not procrastinate when there is no time for procrastination). For an easy interpretation of the notation, we denote each edge of type (1) by $e_\nearrow(v_t^\ell)$ and each edge of type (2) by $e_\rightarrow(v_t^\ell)$, and they represent progression and procrastination of the agent at state v_t^ℓ, respectively. Each edge $e = (u, v) \in E$ has a cost $c(e)$, which represents the *effort* to go from state u to state v.

We note that the family of grid graphs is not a compromise. Kleinberg and Oren [8] showed that all graphs that exploit the worst-case behavior of a time inconsistent agent on general directed acyclic graphs, contain as a minor a graph that is trivially simulated by a grid

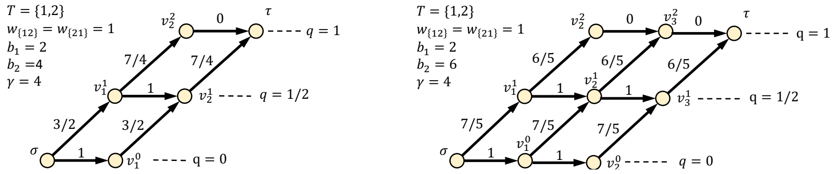


Figure 1 On the left an example task graph where the overall cost of completing the tasks increases compared to the case where the individuals are not aware of each other. On the right, a task graph where the overall cost to complete the tasks decreases.

graph. Specifically, Kleinberg and Oren [8] show that every task graph that forces an agent to follow a path that has exponentially larger cost compared to the optimum path, contains as a minor the graph that has $n+2$ states $\sigma = v_0, v_1, \dots, v_n, \tau$ and edges (v_i, v_{i+1}) and (v_i, τ) for all $0 \le i \le n$. We construct a grid with states $\sigma = v_0^0, v_1^0, \dots, v_n^0, v_1^1, v_2^1, \dots, v_{n+1}^1 = \tau$, and edges (v_i^0, v_{i+1}^0) for all $0 \le i \le n-1$ with cost equal to the cost of the edges (v_i, v_{i+1}) in the worst-case graph; edges (v_i^0, v_{i+1}^1) for $1 \le i \le n$ with cost equal to the cost of the edges (v_i, τ) in the worst-case graph; and edges (v_i^1, v_{i+1}^1) for $0 \le i \le n$ with cost 0. The grid graph essentially splits the state τ of the worst-case graph and replaces it with a path of cost 0 in all of its edges. This path represents completion of the task, as the remaining path to τ is zero.

Given a task graph, present-time biased agents act according to their interpretation of the most effective sequence of actions. Notice that the (objectively) optimum sequence of actions by the agent corresponds to the shortest path in the task graph from σ to τ. An agent who follows the shortest path from a state executes the best actions and minimizes her overall cost. However, according to the quasi-hyperbolic–discounting model [12] the agent misinterprets the cost of her next actions: the costs of all actions in the next step are amplified by a multiplicative factor b. In other words, at state v_t^ℓ the agent perceives the overall effort to accomplish the task as $b \cdot c(e_{\nearrow}(v_t^\ell)) + d(v_{t+1}^{\ell+1}, \tau)$, if she chooses to make progress, and as $b \cdot c(e_{\rightarrow}(v_t^\ell)) + d(v_{t+1}^{\ell}, \tau)$, if she chooses to postpone actions to future time steps. Subsequently, the agent picks the action that minimizes the perceived cost. Throughout the paper we assume that if the perceived cost of making progress equals the perceived cost to procrastinate, the agent chooses to make progress.

In our model we assume that the members of a team T are assigned individual tasks graphs. Each agent (team member) performs on his own task graph. The present-time bias of each agent is affected by two factors, the *personal bias* and the *social bias*. The personal bias $b_i \ge 1$ depends solely on the agent and it does not change throughout the process. As the agents proceed by performing the tasks assigned to them they interact with each other and learn their progress. The unnormalized progress of an agent $i \in T$ at time t is denoted by $r_i(t) \in \{0, \dots, \ell_{\max}\}$, where $\ell_{\max}$ is the maximum progress level on i's task graph. We define also the (normalized) progress $q_i(t) \in [0,1]$ of agent i as $q_i(t) = r_i(t)/\ell_{\max}$. This allows us to compare the progress of agents with different task graphs and different number of progress levels.

Being aware of the progress made by the other members of the team might affect the motivation of an agent. We assume that agents exert to each other an amount of social influence, which is denoted by a weight $w_{ij} \in [0,1]$, for each pair of agents i and j (in general $w_{ij} \ne w_{ji}$). We assume $w_{ii} = 0$, for all agents $i \in T$. We are now ready to define the social-bias factor of our model.

▶ **Definition 1.** *The social bias of an agent $i \in T$ at time t is denoted by $\Gamma_i^T(t)$ and defined as*

$$\Gamma_i^T(t) = \gamma_i^{\sum_{j \in T} w_{ij}(q_i(t) - q_j(t))},$$

where $\gamma_i \geq 1$ is a social-bias parameter, $w_{ij} \in [0,1]$ is the social influence between agents i and j, and $q_j(t)$ is the (normalized) progress level of each agent $j \in T$ at time t.

Obviously one can consider different functions $\Gamma_i^T(t)$, but in this paper we specialize to this particular form. Notice that our model is a generalization of the model by Kleinberg and Oren because $T = \{i\}$ implies that $\Gamma_i^T(t) = 1$. The simplest, nontrivial, case for our model is when there are only two agents, that is, $T = \{i, j\}$. In that case $\Gamma_i^{\{i,j\}}(t) = \gamma_i^{q_i(t) - q_j(t)}$. Often we assume that we have $\gamma_i = \gamma_j$ for all $i, j \in T$ and in this case we just use $\gamma_i = \gamma$. Whenever $\Gamma_i^T(t) < 1$ we say that agent i is motivated because of the influence of j, and when $\Gamma_i^T(t) > 1$ we say that agent i is discouraged. The following property follows from our model.

▶ **Property 2.** *Consider a team T and an agent $i \in T$, with $\gamma_i \geq 1$ and $w_{ij} \geq 0$, for all $j \in T$. If $q_j(t) \geq q_i(t)$ for all $j \in T$ then $\Gamma_i^T(t) \leq 1$. Similarly, if $q_j(t) \leq q_i(t)$ for all $j \in T$ then $\Gamma_i^T(t) \geq 1$.*

The *present-time bias* of an agent $i \in T$ is defined as $B_i^T(t) = \max\{b_i \Gamma_i^T(t), 1\}$. The present-time bias affects the *perceived* cost of a path for an agent. A *path* p is a sequence of nodes $p = \langle v_1, \ldots, v_k \rangle$ such that (v_j, v_{j+1}) is an edge of the graph for all $1 \leq j \leq k-1$. The *cost* of p is $\sum_{j=1}^{k-1} c(v_j, v_{j+1})$. Consider a path $p = \langle v_1, \ldots, v_k \rangle$, where $v_1 = v_t^\ell$ is the current state of an agent $i \in T$. Then the *perceived cost* for an agent i for the path p is

$$B_i^T(t) \cdot c(v_1, v_2) + d(v_2, \tau).$$

An agent, who aims to complete the assigned task G, follows a path from σ to τ. Note that each such path is of the form $p = \langle \sigma = v_0^0, v_1^{\ell_1}, \ldots, v_{t_{\max}}^{\ell_{\max}} = \tau \rangle$. We call such a path *progress path* on task G. Given two progress paths $p = \langle \sigma = v_0^0, v_1^{\ell_1}, v_2^{\ell_2}, \ldots, v_{t_{\max}}^{\ell_{\max}} = \tau \rangle$ and $p' = \langle \sigma = v_0^0, v_1^{\ell'_1}, v_2^{\ell'_2}, \ldots, v_{t_{\max}}^{\ell_{\max}} = \tau \rangle$ on the same task G, we say that p *is above* path p', and write $p \succeq_G p'$, if for each $t = 1, \ldots, t_{\max}$ we have that $\ell_t \geq \ell'_t$. We define $\preceq_G$ (i.e., *below*) analogously. Let $p^T(i)$ be the progress path of agent $i \in T$. We use $p(i)$ to denote $p^{\{i\}}(i)$.

Example 1.[1] We demonstrate our model with two examples in the case of teams with two members. In the first example, in Figure 1 (left), the total cost of the progress paths followed by the two agents increases when the two agents interact compared to the case where the two agents do not have knowledge of each other's progress. The two agents operate on the same task graph. Agent 1 has personal bias $b_1 = 2$ and agent 2 has $b_2 = 4$. If agent 1 would operate independently of agent 2, it would start from state $\sigma = v_0^0$, and would evaluate the options of following either edge $e_\nearrow(v_0^0)$ (i.e., to make progress) or edge $e_\rightarrow(v_0^0)$ (i.e., procrastinate). The perceived cost of following edge $e_\nearrow(v_0^0)$ is $\Gamma_1(0) \cdot b_1 \cdot c(e_\nearrow(v_0^0)) + d(v_1^1, \tau) = 19/4$, whereas the perceived cost of following the edge $e_\rightarrow(v_0^0)$ is $\Gamma_1(0) \cdot b_1 \cdot c(e_\rightarrow(v_0^0)) + d(v_1^0, \tau) = 21/4$. Therefore, at state σ agent 1 would follow edge $e_\nearrow(v_0^0)$. At state v_1^1 the agent would again proceed based on the perceived cost of following either edge $e_\nearrow(v_1^1)$ or $e_\rightarrow(v_1^1)$. The perceived cost of following edge $e_\nearrow(v_1^1)$ is $\Gamma_1(1) \cdot b_1 \cdot c(e_\nearrow(v_1^1)) + d(v_2^2, \tau) = 14/4$, whereas the perceived cost of following the edge $e_\rightarrow(v_1^1)$ is $\Gamma_1(1) \cdot b_1 \cdot c(e_\rightarrow(v_1^1)) + d(v_2^1, \tau) = 15/4$. Therefore, at

[1] In the appendix we provide the examples with the calculations performed explicitly.

state v_1^1 agent 1 would follow edge $e_{\nearrow}(v_1^1)$. For the last edge, the agent has only the option to follow the edge $e_{\rightarrow}(v_2^2)$ with cost 0 to reach τ. Hence, the cost of the progress path of agent 1 when operating individually would be 13/4.

Similarly, if agent 2 operates independently of agent 1, the perceived cost of following edge $e_{\nearrow}(v_0^0)$ is $\Gamma_2(0) \cdot b_2 \cdot c(e_{\nearrow}(v_0^0)) + d(v_1^1, \tau) = 31/4$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_0^0)$ is $\Gamma_2(0) \cdot b_2 \cdot c(e_{\rightarrow}(v_0^0)) + d(v_1^0, \tau) = 29/4$. Therefore, at state v_0^0 agent 2 would follow edge $e_{\rightarrow}(v_0^0)$. At state v_1^0 the agent has no other options that to follow the edges $e_{\nearrow}(v_1^0)$ and then the edge $e_{\nearrow}(v_2^1)$ to reach τ. Hence, the cost of the progress path of agent 2 when operating individually would be 17/4.

Now we analyze the behavior of agents 1 and 2 when they collaborate on the same project and they both have to perform the same task. At time step $t = 0$, we have that the social bias is $\Gamma_1^{\{1,2\}}(0) = \gamma^{q_1(0)-q_2(0)} = 4^0 = 1$. Analogously, we have that $\Gamma_2^{\{1,2\}}(0) = 1$. Therefore, the choice of each agent at time step $t = 0$ is the same as when they perform independently as their personal bias remains unchanged. That is, agent 1 follows the edge $e_{\nearrow}(v_0^0)$ making progress 1 at time $t = 1$ and agent 2 follows the edge $e_{\rightarrow}(v_0^0)$ making no progress at time $t = 1$. At time step $t = 1$, agent 1 evaluates the options of following edge $e_{\nearrow}(v_1^1)$ or $e_{\rightarrow}(v_1^1)$. Notice that now the social bias of agent 1 is $\Gamma_1^{\{1,2\}}(1) = \gamma^{q_1(1)-q_2(1)} = 4^{1/2} = 2$. Hence, the perceived cost of following edge $e_{\nearrow}(v_1^1)$ is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2, \tau) = 28/4$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_1^1)$ is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1, \tau) = 23/4$. Therefore, at state v_1^1 agent 1 would follow edge $e_{\rightarrow}(v_1^1)$. For the last edge, agent 1 has only one option, that is, to follow the edge $e_{\nearrow}(v_2^1)$ to reach τ. Hence, the cost of the progress path of agent 1 is 17/4, compared to the cost 13/4 of the progress path that it would follow independently. The progress path of agent 2 does not change when operating with agent 1, as after the first choice to follow edge $e_{\rightarrow}(v_0^0)$ there are not alternative paths that agent 2 could follow. In conclusion, the total cost of the two agents when operating together is 34/4 compared to the 30/4 when operating independently.

Example 2. We now proceed with an example where the collaboration of two agents leads to a decrease to the total cost of their progress paths. Consider the case where two agents 1, 2 with personal biases $b_1 = 2, b_2 = 6$ operate on the task graph in Figure 1 (right). It can be verified that the progress path of agent 1 is $p(i) = \langle \sigma = v_0^0, v_1^1, v_2^2, v_3^2, v_4^2 = \tau \rangle$ and the progress path of agent 2 is $p(2) = \langle \sigma = v_0^0, v_1^0, v_2^0, v_3^1, v_4^2 = \tau \rangle$. Therefore, the total cost of the $p(1)$ and $p(2)$ is 36/5.

Now we consider the case where the two agents interact with each other. Similarly to the first example, at time step $t = 0$ the social bias is $\Gamma_1^{\{1,2\}}(0) = \Gamma_2^{\{1,2\}}(0) = 1$ and hence the choices at time $t = 0$ of agents 1, 2 are the same as in the case where they operate independently. That is, $q_1(1) = 1/2$ and $q_2(1) = 0$. At time step $t = 1$, the social bias of agent 1 is $\Gamma_1^{\{1,2\}}(1) = \gamma^{q_1(1)-q_2(1)} = 4^{1/2} = 2$. According to our model, the perceived cost of following edge $e_{\nearrow}(v_1^1)$ by agent 1 is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2, \tau) = 24/5$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_1^1)$ is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1, \tau) = 26/5$. Therefore, at state v_1^1 agent 1 follows edge $e_{\nearrow}(v_1^1)$. We now review the decision of agent 2 at time $t = 1$, whose social bias is $\Gamma_2^{\{1,2\}}(1) = \gamma^{q_2(1)-q_1(1)} = 4^{-1/2} = 1/2$. Hence, agent 2 at time $t = 1$ perceives cost $\Gamma_2^{\{1,2\}}(1) \cdot b_2 \cdot c(e_{\nearrow}(v_1^0)) + d(v_2^1, \tau) = 27/5$ for following edge $e_{\nearrow}(v_1^0)$, and cost $\Gamma_2^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^0)) + d(v_2^0, \tau) = 28/5$ for following edge $e_{\rightarrow}(v_1^0)$. Therefore, at state v_1^0 agent 2 follows edge $e_{\nearrow}(v_1^0)$ to reach state v_2^1. At time $t = 2$, we have $q_1(2) = 1, q_2(2) = 1/2$. Agent 1 has no options other than to follow the path $\langle v_2^2, v_3^2, \tau \rangle$ to reach τ. The social bias of agent 2 at time $t = 2$ is $\Gamma_2^{\{1,2\}}(2) = \gamma^{q_2(2)-q_1(2)} = 1/2$. Hence, the perceived cost of agent 2 at

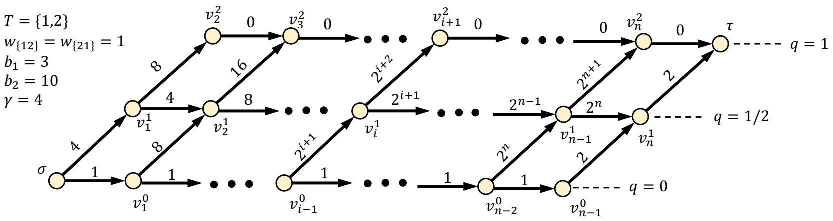


Figure 2 An example where the interaction of two agents increases the total cost of the two progress paths exponentially, even when they operate on the same task graph. When the agents operate independently, they follow the progress paths $p(1) = \langle \sigma, v_1^1, v_2^2, v_3^2, \ldots, \tau \rangle$, $p(2) = \langle \sigma, v_1^0, \ldots, v_{n-1}^0, v_n^1, \tau \rangle$, with total cost $\Theta(n)$. When the agents collaborate, they follow the progress paths $p^{\{1,2\}}(1) = \langle \sigma, v_1^1, v_2^1, \ldots, v_n^1, \tau \rangle$, $p^{\{1,2\}}(2) = \langle \sigma, v_1^0, \ldots, v_{n-1}^0, v_n^1, \tau \rangle$, with total cost $\Theta(2^n)$.

time $t = 2$ in the case of following the edge $e_{\nearrow}(v_2^1)$ is $\Gamma_2^{\{1,2\}}(2) \cdot b_2 \cdot c(e_{\nearrow}(v_2^1)) + d(v_3^2, \tau) = 18/5$, and in the case of following the edge $e_{\rightarrow}(v_2^1)$ is $\Gamma_2^{\{1,2\}}(2) \cdot b_2 \cdot c(e_{\rightarrow}(v_2^1)) + d(v_3^1, \tau) = 21/5$. Therefore, at state v_2^1 agent 2 follows edge $e_{\nearrow}(v_2^1)$. Finally, at time $t = 3$, both agents have no other option than to follow edge (v_3^2, τ) to reach τ. In conclusion, agent 1 follows the progress path $p^{\{1,2\}}(1) = \langle \sigma = v_0^0, v_1^1, v_2^2, v_3^2, v_4^2 = \tau \rangle$ and agent 2 follows that progress path $p^{\{1,2\}}(2) = \langle \sigma = v_0^0, v_1^0, v_2^1, v_3^2, v_4^2 = \tau \rangle$, with total cost 31/5. That is, if agents 1, 2 collaborate they decrease the total cost.

▶ **Corollary 3.** *The total cost of the progress paths of a team can either decrease or increase (or, of course, remain the same) compared to the total cost of the progress paths of the team members when they operate in isolation (i.e., with no communication) on the same tasks.*

3 Limitations and Further Assumptions

We now show that the vanilla version of our model can lead to unnatural phenomena in the interaction of the agents in a team. We construct examples having two interacting agents. Guided by these extreme behaviors we make a set of reasonable assumptions that eliminate those unnatural phenomena. Similar assumptions have been made previously for the behavior of individual agents in absence of a team. More specifically, Gravin et al. [7] showed that the progress path of a time-inconsistent agent can have exponentially larger cost compared to the optimal progress path on a task graph. Here, we extend their example to show that there can be an exponential increase to the total cost of the progress paths of two agents, compared to the case where they operate individually. Our example is depicted in Figure 2. We note that unlike Gravin et al. [7] , where the cost of an agent can be exponentially larger compared to the optimal progress path, which was never an option of the agent, in our case the increase in the total cost is compared to the progress path in the case where the agents operate individually.

The main reason behind the exponential increase in the total cost of two agents is that the optimal cost of completing the task can increase at a future state in the progress path of an agent. In many scenarios, this is an unnatural phenomenon as it implies that the required effort to complete a task increases exponentially over time. Gravin et al. [7] introduce the following two assumptions on the task graph that eliminate such pathological instances.

▶ **Property 4** (Bounded-distance property)**.** *Let G be a task graph. For every vertex $v \in V(G)$, it is $d(v, \tau) \leq d(\sigma, \tau)$.*

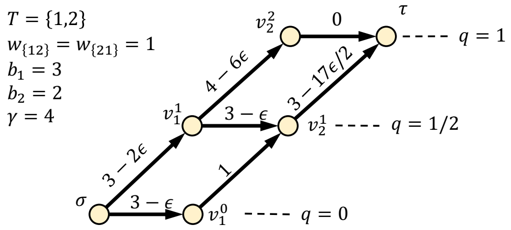


Figure 3 An example where the motivator affects the procrastinator to procrastinate further.

The bounded-distance property allows only task graphs in which the optimal path to complete the task from any state is never worse than the optimal progress path from the initial state. This is a natural assumption in a plethora of real-world tasks. For instance, this includes the tasks in which starting over is always a free and feasible option. Gravin et al. [7] show that for task graph with the bounded-distance property the cost of an agent increases by at most a factor of $\mathcal{O}(n)$, compared to the optimal progress path.

▶ **Property 5** (Monotone-distance property). *For every transition from a vertex u to a vertex v, where $u, v \in V(G)$, it holds that $d(v, \tau) \leq d(u, \tau)$.*

The monotone-distance property of task graphs implies that the cost to complete the task does not increase over time. Notice that the monotone-distance property implies the bounded-distance property: any graph with the monotone-distance property also has the bounded-distance property. Gravin et al. [7] show that if the task graph has the monotone-distance property, and the present-time bias of the agent is drawn from a restricted distribution, then the cost of the progress path compared to the optimal progress path is bounded by a factor much smaller than n.

In Section 4 we study the behavior of agents in task graphs that obey Properties 4 and 5. More specifically, we show that the total cost of progress paths by all agents cannot increase more than a factor of $\mathcal{O}(n)$, compared to the cost of the progress paths in the case where the agents operate individually. We further show that this bound is tight.

In our model, we consider the interaction of two or more agents, which introduces further pathological scenarios in the behavior of the agents. For instance, consider the example in Figure 3, where both agents in a team $T = \{1, 2\}$ operate on the same task graph. Agent 1 has a higher bias than agent 2, and follows the optimal progress path (that is, the path $p(1) = \langle \sigma = v_0^0, v_1^0, v_2^1, v_3^2 = \tau \rangle$ with cost $7 - 9.5\epsilon$), while the agent with lower bias follows a progress path with larger cost (that is, the path $p(2) = \langle \sigma = v_0^0, v_1^1, v_2^2, v_3^3 = \tau \rangle$ with cost $7 - 8\epsilon$). This phenomenon is unnatural as in this example the motivated individual (i.e., the agent with smaller personal bias) follows a progress path with larger cost compared to the procrastinating individual (i.e., the agent with larger personal bias). Moreover, when the two agents in Figure 3 interact, the motivated individual causes the procrastinating individual to further procrastinate (follow a progress path with larger cost). To eliminate such behaviors, we introduce an additional assumption on the task graph. Our assumption is that from any state the action leading to progress costs more than the action of postponing the progress.

▶ **Property 6.** *Given a task graph $G = (V, E)$ it holds that $c(e_{\nearrow}(v_t^\ell)) \geq c(e_{\rightarrow}(v_t^\ell))$, for all $v_t^\ell \in V$.*

4 Team Behavior on Task Graphs

We now study the behavior of time-inconsistent agents in teams under our model. We begin by bounding the change in the total cost of all progress paths compared to the cost in the case where the agents operate individually. The objective is to bound the maximum loss on the total effort made by the team when the agents communicate their progress, compared to the case where the agents operate individually.[2]

▶ **Lemma 7.** *Let $T = \{i_1, \ldots, i_k\}$ be a team of agents, where $w_{ij} = 1$, for all $i, j \in T$, operating on task graphs $G_1, \ldots, G_k$, where all task graphs should be completed in $t_{\max} = n$ time steps and all task graphs have Properties 4 and 5. The scenario in which the agents collaborate can lead to total cost of their progress paths that is larger than the case where they operate individually by a factor $\Omega(n)$.*

Gravin et al. [7] showed that the cost of an agent on a graph with Properties 4 and 5 cannot exceed n times the cost of the shortest path. Their proof suffices to prove the following lemma.

▶ **Lemma 8.** *Let $T = \{i_1, \ldots, i_k\}$ be a team of agents, where $w_{ij} = 1$, for all $i, j \in T$, operating on task graphs $G_1, \ldots, G_k$, where all task graphs have $t_{\max} = n$ time steps and all task graphs have Properties 4 and 5. Collaboration can increase by at most a factor of n the total cost spent by the agents to accomplish the assignment compared to the total cost of the agents operating individually.*

We now provide a lower bound on the speedup that the collaboration in a team can achieve.

▶ **Lemma 9.** *Let $T = \{i_1, \ldots, i_k\}$ be a team of agents, where $w_{ij} = 1$, for all $i, j \in T$, operating on task graphs $G_1, \ldots, G_k$, where all task graphs should be completed in $t_{\max} = n$ time steps and all task graphs obey Properties 4, 5 and 6. The total cost may decrease by a factor of $\Omega(n)$ due to collaboration.*

Agents operating on identical task graphs. We now compare the progress paths of the agents and the way they relate to each other, in the case where all agents perform on the same task graph. We begin with the following lemma that states that the order of the progress paths of the agents is the same as the reverse order of their personal biases.

▶ **Lemma 10.** *Consider two agent $i, j \in T$ operating on the same task graph G. If $b_i \geq b_j$, then $p(i) \preceq_G p(j)$.*

Next we relate all progress paths when the agents collaborate with respect to the progress paths of the agents with the maximum and minimum personal biases. That is, throughout the process of collaboration in a team, no agent does more (resp., less) progress than the most (resp., least) motivated agent does independently, at any time. The lemma suggests that all progress paths in the case where the agents collaborate are between the progress paths of the most motivated and the least motivated agents when operating individually. We call this the *envelope property.*

▶ **Lemma 11** (Envelope property). *Consider a team with agents $T = \{i_1, \ldots, i_k\}$, with $b_{i_1} \geq \cdots \geq b_{i_k}$, operating over the same task graph G. For each $i \in T$ we have that $p^T(i) \succeq_G p(i_1)$ and $p^T(i) \preceq_G p(i_k)$.*

[2] The proof of this and further results appear in the appendix.

We now have developed a better understanding on the interaction between agents of a team operating on the same task graph. Next, we use these results to further bound the ratio of the total cost of progress paths when the team collaborates compared to the cost of operating individually. We observe that this setting still allows examples in which the cost can increase by a factor of $\Omega(n)$ (as in Figure 4, used to prove Lemma 7). To cope with such extreme examples, we introduce the following restriction on the task graph on which the team operates.

▶ **Property 12.** *For every two vertices $v_i^j, v_i^l \in V(G)$, where $j \leq l$, it holds that $c(v_i^j, v_{i+1}^j) \geq c(v_i^l, v_{i+1}^l)$.*

Essentially, Property 12 implies that procrastinating at a specific time step cannot cost more if the agent made more progress compared to the case where the agent made less progress at the same time step. This is a reasonable restriction to the structure of the task graph. We acknowledge, however, that there exist scenarios where Property 12 is not natural. For instance, such a scenario appears in the case of lab experiments where the procrastination of an agent after starting the experiment might lead to a waste of the whole experiment (i.e., the resources), while postponing the starting time of the experiment simply delays the whole process.

▶ **Lemma 13.** *Let T be a team of k agents operating on the same task graph, which has Properties 4, 5, 6, and 12. The total cost of all progress paths when the agents collaborate is at most k times higher than the sum of cost of progress paths when the agent operate independently.*

5 Assignment Problems

Until now we studied the scenario where the assignment of task graphs to agents is given in advance. Another natural scenario is when a given task can be assigned to more than one agents with similar skills. Can we then determine the best assignment so as to minimize the cost due to procrastination? A simple special case is when a project consists of $n/2$ identical tasks and there exist n agents which should be grouped into $n/2$ two-member teams, such that the total cost payed by all agents is minimized.

▶ **Problem 14.** *Assume that we are given $n/2$ copies of the same task graph G, n agents $1, \ldots, n$ with personal biases $b_1, \ldots, b_n$ and social-influence weights w_{ij} for all $1 \leq i, j \leq n, i \neq j$. The goal is to partition the n agents into $n/2$ two-member teams, such that when the two agents of each team work on the common task specified by G, the total cost over all agents is minimized.*

Problem 14 has a simple solution. For each of the $\binom{n}{2}$ pairs of agents, we can compute the cost of the two agents collaborating together on the given task. We obtain a complete weighted graph where each node represents an agent and each edge weights correspond to the total cost of the two agents paired as a team. The problem then reduces to finding a minimum-weight matching.

Assume that we have a single team of n agents, one project consisting of n tasks, each one having its own task graph, and we need to assign one agent to each of the tasks, so as to minimize the total cost of finishing the project. Without much loss of generality we assume that each team member can perform all tasks. The optimal assignment problem takes the following form.

▶ **Problem 15** (OptimalGroupAssignment). *Consider n task graphs $G_1, \dots, G_n$, n agents $1, \dots, n$ with personal biases $b_1, \dots, b_n$ and social-influence weights w_{ij} for all $1 \leq i, j \leq n, i \neq j$. The task is to assign one agent to each task, such that the total cost of completing all the tasks in the collaborative-procrastination model is minimized.*

We next prove that this problem is hard.

▶ **Theorem 16.** *The OptimalGroupAssignment problem is* **NP**-*hard.*

6 Related Work

Some of the first studies in economics attempting to formulate time-inconsistent planning behavior was the work of Strotz [14] and Pollak [13]. The theory of time-inconsistency developed to what is called *quasi-hyperbolic discounting* Laibson [12], Frederick et al. [5]. The theory provides a natural way to model the decision of an agent to procrastinate, using the notion of *present-time bias* – the tendency to view costs and benefits that are incurred at the present moment to be more salient than those incurred in the future. Kleinberg and Oren [8] propose a graph-theoretic model, in which dependencies among actions are represented by a directed acyclic graph, and a time-inconsistent agent follows a path through this graph based on the agent's biased evaluation of the actions at each time step. Kleinberg and Oren [8] characterize the worst-case procrastination ratio, and they consider the problem of reducing the procrastination cost by deleting nodes and/or edges from an underlying graph.

Gravin et al. [7] consider the case where the present-time bias of the agent is drawn at each time step at random, from a distribution $\mathcal{F}$, They characterize the worst possible cost of a path chosen by an agent compared to the cost of the optimal path, and under reasonable assumptions they provide bounds for this ratio. Kleinberg et al. [9] model the behavior of sophisticated agents – agents who are aware of their tendency to procrastinate and they plan in advance. Their study includes tight upper bounds on the procrastination relatively to the optimal path in a task graph. Kleinberg et al. [10] consider the interaction of multiple biases on an agent's behavior: they study the interaction of present-time bias factor and sunk-cost bias factor – the tendency to incorporate costs incurred in the past into ones plans for the future, even when these past costs are no longer relevant to optimal planning. Moreover, based again on the model of Kleinberg and Oren [8], several studies consider optimization problems where the objective is to minimize the cost of the path followed by an agent [2, 4, 3].

Gans and Landry [6] consider the interaction of teams with two present-time biased agents who collaborate to accomplish a common goal. They assume that both agents can accomplish all subtasks, and that the agents can either be sophisticated or be naïve – in the sense that they either know their present-time bias factor or not. The objective of each agent is to complete the task with the minimum possible effort from their side. The model of Gans and Landry [6] is different from ours as progress can be done by any agent, and at each time step there is no distinction with respect to which agent achieved the progress in the previous step.

7 Conclusion and Open Problems

In this paper we extended the model of Kleinberg and Oren [8] on time-inconsistent planning into settings where individuals are members of a team and the decision on whether to perform a task or postpone it depends on the progress of the other team members. Our model incorporates phenomena that are encountered in real life: participating in a team can motivate (or demotivate) individuals compared to when they work individually. In the proposed setting we showed how different assumptions allow to deduce the extent that participation to a team may increase or decrease performance. We also showed that our model can be used to define matching and team-formation problems, when the goal is to form teams that keep the members motivated.

Whereas our model captures some elements of how agents in teams may collaborate, there are many other modeling choices. Often, the load of one member who has not progressed may be transferred to other team members; this may lead to free-riding phenomena, and a game-theoretic approach may be suitable to model such settings. Note that Lemma 11 implies that the effort of a member who participates on a team cannot exceed the one of the most efficient member; in particular, it implies that the most efficient member cannot improve by participating in a team. Often this is not the case: for instance, one can attempt to model *competition* between team members, which may lead to more efficient performance.

References

1 George A Akerlof. Procrastination and obedience. *The American Economic Review*, 81(2):1–19, 1991.

2 Susanne Albers and Dennis Kraft. Motivating time-inconsistent agents: A computational approach. In *Web and Internet Economics*, pages 309–323, 2016.

3 Susanne Albers and Dennis Kraft. On the Value of Penalties in Time-Inconsistent Planning. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, pages 10:1–10:12, 2017.

4 Susanne Albers and Dennis Kraft. The price of uncertainty in present-biased planning. In *Web and Internet Economics*, pages 325–339, 2017.

5 Shane Frederick, George Loewenstein, and Ted O'donoghue. Time discounting and time preference: A critical review. *Journal of economic literature*, 40(2):351–401, 2002.

6 Joshua S Gans and Peter Landry. Procrastination in teams. Technical report, National Bureau of Economic Research, 2016.

7 Nick Gravin, Nicole Immorlica, Brendan Lucier, and Emmanouil Pountourakis. Procrastination with variable present bias. In *Proc. of the 2016 ACM Conference on Economics and Computation*, EC '16. ACM, 2016.

8 Jon Kleinberg and Sigal Oren. Time-inconsistent planning: a computational problem in behavioral economics. In *Proc. of the fifteenth ACM conference on Economics and computation*, EC '14, pages 547–564. ACM, 2014.

9 Jon Kleinberg, Sigal Oren, and Manish Raghavan. Planning problems for sophisticated agents with present bias. In *Proc. of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 343–360, New York, NY, USA, 2016. ACM. doi:10.1145/2940716.2940764.

10 Jon Kleinberg, Sigal Oren, and Manish Raghavan. Planning with multiple biases. *arXiv preprint arXiv:1706.01062*, 2017.

11 Jon Kleinberg and Maithra Raghu. Team performance with test scores. In *Proc. of the Sixteenth ACM Conference on Economics and Computation*, pages 511–528. ACM, 2015.

12 David Laibson. Golden eggs and hyperbolic discounting. *The Quarterly Journal of Economics*, 112(2):443–478, 1997.

13 Robert A Pollak. Consistent planning. *The Review of Economic Studies*, 35(2):201–208, 1968.

14 Robert Henry Strotz. Myopia and inconsistency in dynamic utility maximization. *The Review of Economic Studies*, 23(3):165–180, 1955.

15 Stefan Wuchty, Benjamin F Jones, and Brian Uzzi. The increasing dominance of teams in production of knowledge. *Science*, 316(5827):1036–1039, 2007.

A Appendix

Examples of Section 2 with Calculations

In this section we provide the calculations used for the examples in Section 2.

Example 1. We demonstrate our model with two examples in the case of teams with two members. In the first example, in Figure 1 (left), the total cost of the progress paths followed by the two agents increases when the two agents interact compared to the case where the two agents do not have knowledge of each other's progress. The two agents operate on the same task graph. Agent 1 has personal bias $b_1 = 2$ and agent 2 has $b_2 = 4$. If agent 1 would operate independently of agent 2, it would start from state $\sigma = v_0^0$, and would evaluate the options of following either edge $e_{\nearrow}(v_0^0)$ (i.e., to make progress) or edge $e_{\rightarrow}(v_0^0)$ (i.e., procrastinate). The perceived cost of following edge $e_{\nearrow}(v_0^0)$ is $\Gamma_1(0) \cdot b_1 \cdot c(e_{\nearrow}(v_0^0)) + d(v_1^1, \tau) = 1 \cdot 2 \cdot \frac{3}{2} + 7/4 + 0 = \frac{19}{4}$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_0^0)$ is $\Gamma_1(0) \cdot b_1 \cdot c(e_{\rightarrow}(v_0^0)) + d(v_1^0, \tau) = 1 \cdot 2 \cdot 1 + \frac{3}{2} + \frac{7}{4} = \frac{21}{4}$. Therefore, at state σ agent 1 would follow edge $e_{\nearrow}(v_0^0)$. At state v_1^1 the agent would again proceed based on the perceived cost of following either edge $e_{\nearrow}(v_1^1)$ or $e_{\rightarrow}(v_1^1)$. The perceived cost of following edge $e_{\nearrow}(v_1^1)$ is $\Gamma_1(1) \cdot b_1 \cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2, \tau) = 1 \cdot 2 \cdot \frac{7}{4} + 0 = \frac{14}{4}$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_1^1)$ is $\Gamma_1(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1, \tau) = 1 \cdot 2 \cdot 1 + \frac{7}{4} = \frac{15}{4}$. Therefore, at state v_1^1 agent 1 would follow edge $e_{\nearrow}(v_1^1)$. For the last edge, the agent has only the option to follow the edge $e_{\rightarrow}(v_2^2)$ with cost 0 to reach τ. Hence, the cost of the progress path of agent 1 when operating individually would be $\frac{3}{2} + \frac{7}{4} + 0 = \frac{13}{4}$.

Similarly, if agent 2 operates independently of agent 1, the perceived cost of following edge $e_{\nearrow}(v_0^0)$ is $\Gamma_2(0) \cdot b_2 \cdot c(e_{\nearrow}(v_0^0)) + d(v_1^1, \tau) = 1 \cdot 4 \cdot \frac{3}{2} + \frac{7}{4} + 0 = \frac{31}{4}$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_0^0)$ is $\Gamma_2(0) \cdot b_2 \cdot c(e_{\rightarrow}(v_0^0)) + d(v_1^0, \tau) = 1 \cdot 4 \cdot 1 + \frac{3}{2} + \frac{7}{4} = \frac{29}{4}$. Therefore, at state v_0^0 agent 2 would follow edge $e_{\rightarrow}(v_0^0)$. At state v_1^0 the agent has no other options that to follow the edges $e_{\nearrow}(v_1^0)$ and then the edge $e_{\nearrow}(v_2^1)$ to reach τ. Hence, the cost of the progress path of agent 2 when operating individually would be $1 + \frac{3}{2} + \frac{7}{4} = \frac{17}{4}$.

Now we analyze the behavior of agents 1 and 2 when they collaborate on the same project and they both have to perform the same task. At time step $t = 0$, we have that the social bias is $\Gamma_1^{\{1,2\}}(0) = \gamma^{q_1(0) - q_2(0)} = 4^0 = 1$. Analogously, we have that $\Gamma_2^{\{1,2\}}(0) = 1$. Therefore, the choice of each agent at time step $t = 0$ is the same as when they perform independently as their personal bias remains unchanged. That is, agent 1 follows the edge $e_{\nearrow}(v_0^0)$ making progress 1 at time $t = 1$ and agent 2 follows the edge $e_{\rightarrow}(v_0^0)$ making no progress at time $t = 1$. At time step $t = 1$, agent 1 evaluates the options of following edge $e_{\nearrow}(v_1^1)$ or $e_{\rightarrow}(v_1^1)$. Notice that now the social bias of agent 1 is $\Gamma_1^{\{1,2\}}(1) = \gamma^{q_1(1) - q_2(1)} = 4^{1/2} = 2$. Hence, the perceived cost of following edge $e_{\nearrow}(v_1^1)$ is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2, \tau) = 2 \cdot 2 \cdot \frac{7}{4} + 0 = \frac{28}{4}$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_1^1)$ is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1, \tau) = 2 \cdot 2 \cdot 1 + \frac{7}{4} = \frac{23}{4}$. Therefore, at state v_1^1 agent 1 would follow edge $e_{\rightarrow}(v_1^1)$. For the last edge, agent 1 has only one option, that is, to follow the edge $e_{\nearrow}(v_2^1)$ to reach τ. Hence, the cost of the progress path of agent 1 is $\frac{3}{2} + 1 + \frac{7}{4} = \frac{17}{4}$, compared to the cost 13/4 of the progress path that it would follow independently. The progress path of agent 2 does not change when operating with agent 1, as after the first choice to follow edge $e_{\rightarrow}(v_0^0)$ there are not alternative paths that agent 2 could follow. In conclusion, the total cost of the two agents when operating together is 34/4 compared to the 30/4 when operating independently.

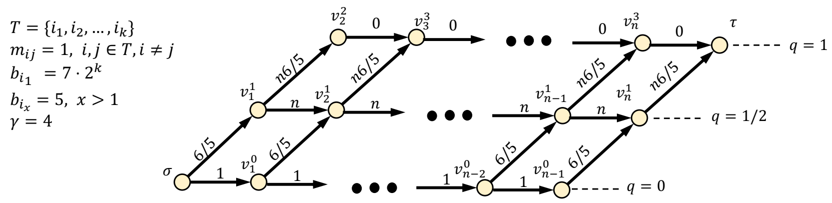


Figure 4 An example where the collaboration leads to an increase, by an $\mathcal{O}(n)$ factor, on the total cost.

Example 2. We now proceed with an example where the collaboration of two agents leads to a decrease to the total cost of their progress paths. Consider the case where two agents $1, 2$ with personal biases $b_1 = 2, b_2 = 6$ operate on the task graph in Figure 1 (right). It can be verified that the progress path of agent 1 is $p(i) = \langle \sigma = v_0^0, v_1^1, v_2^2, v_3^2, v_4^2 = \tau \rangle$ and the progress path of agent 2 is $p(2) = \langle \sigma = v_0^0, v_1^0, v_2^0, v_3^1, v_4^2 = \tau \rangle$. Therefore, the total cost of the $p(1)$ and $p(2)$ is $\frac{36}{5}$.

Now we consider the case where the two agents interact with each other. Similarly to the first example, at time step $t = 0$ the social bias is $\Gamma_1^{\{1,2\}}(0) = \Gamma_2^{\{1,2\}}(0) = 1$ and hence the choices at time $t = 0$ of agents $1, 2$ are the same as in the case where they operate independently. That is, $q_1(1) = 1/2$ and $q_2(1) = 0$. At time step $t = 1$, the social bias of agent 1 is $\Gamma_1^{\{1,2\}}(1) = \gamma^{q_1(1) - q_2(1)} = 4^{1/2} = 2$. According to our model, the perceived cost of following edge $e_{\nearrow}(v_1^1)$ by agent 1 is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2, \tau) = 2 \cdot 2 \cdot \frac{6}{5} + 0 + 0 = \frac{24}{5}$, whereas the perceived cost of following the edge $e_{\rightarrow}(v_1^1)$ is $\Gamma_1^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1, \tau) = 2 \cdot 2 \cdot 1 + \frac{6}{5} + 0 = \frac{26}{5}$. Therefore, at state v_1^1 agent 1 follows edge $e_{\nearrow}(v_1^1)$. We now review the decision of agent 2 at time $t = 1$, whose social bias is $\Gamma_2^{\{1,2\}}(1) = \gamma^{q_2(1) - q_1(1)} = 4^{-1/2} = 1/2$. Hence, agent 2 at time $t = 1$ perceives cost $\Gamma_2^{\{1,2\}}(1) \cdot b_2 \cdot c(e_{\nearrow}(v_1^0)) + d(v_2^1, \tau) = \frac{1}{2} \cdot 6 \cdot \frac{7}{5} + \frac{6}{5} + 0 = \frac{27}{5}$ for following edge $e_{\nearrow}(v_1^0)$, and cost $\Gamma_2^{\{1,2\}}(1) \cdot b_1 \cdot c(e_{\rightarrow}(v_1^0)) + d(v_2^0, \tau) = \frac{1}{2} \cdot 6 \cdot 1 + \frac{7}{5} + \frac{6}{5} = \frac{28}{5}$ for following edge $e_{\rightarrow}(v_1^0)$. Therefore, at state v_1^0 agent 2 follows edge $e_{\nearrow}(v_1^0)$ to reach state v_2^1. At time $t = 2$, we have $q_1(2) = 1, q_2(2) = 1/2$. Agent 1 has no options other than to follow the path $\langle v_2^2, v_3^2, \tau \rangle$ to reach τ. The social bias of agent 2 at time $t = 2$ is $\Gamma_2^{\{1,2\}}(2) = \gamma^{q_2(2) - q_1(2)} = 4^{-1/2} = 1/2$. Hence, the perceived cost of agent 2 at time $t = 2$ in the case of following the edge $e_{\nearrow}(v_2^1)$ is $\Gamma_2^{\{1,2\}}(2) \cdot b_2 \cdot c(e_{\nearrow}(v_2^1)) + d(v_3^2, \tau) = \frac{1}{2} \cdot 6 \cdot \frac{6}{5} + 0 = \frac{18}{5}$, and in the case of following the edge $e_{\rightarrow}(v_2^1)$ is $\Gamma_2^{\{1,2\}}(2) \cdot b_2 \cdot c(e_{\rightarrow}(v_2^1)) + d(v_3^1, \tau) = \frac{1}{2} \cdot 6 \cdot 1 + \frac{6}{5} = \frac{21}{5}$. Therefore, at state v_2^1 agent 2 follows edge $e_{\nearrow}(v_2^1)$. Finally, at time $t = 3$, both agents have no other option than to follow edge (v_3^2, τ) to reach τ. In conclusion, agent 1 follows the progress path $p^{\{1,2\}}(1) = \langle \sigma = v_0^0, v_1^1, v_2^2, v_3^2, v_4^2 = \tau \rangle$ and agent 2 follows that progress path $p^{\{1,2\}}(2) = \langle \sigma = v_0^0, v_1^0, v_2^1, v_3^2, v_4^2 = \tau \rangle$, with total cost $31/5$. That is, if agents $1, 2$ collaborate they decrease the total cost.

Proofs

Detailed proof of Lemma 7. See Figure 4, where all agents operate on the same task graph. For agent i_1 at state v_0^0 the perceived cost is

$$\begin{aligned} B_{i_1}^T(0) \cdot c(e_{\nearrow}(v_0^0)) + d(v_1^1, \tau) &= b_{i_1} \cdot \Gamma_{i_1}^T(0) \cdot \frac{6}{5} + \frac{6n}{5} \\ &= 7 \cdot 2^k \cdot 4^0 \cdot \frac{6}{5} + \frac{6n}{5} = \frac{42 \cdot 2^k + 6n}{5} \end{aligned}$$

for following edge $e_{\nearrow}(v_0^0)$ and

$$\begin{aligned}B_{i_1}^T(0)\cdot c(e_{\rightarrow}(v_0^0)) + d(v_0^1,\tau) &= b_{i_1} \cdot \Gamma_{i_1}^T(0) + \frac{6}{5} + \frac{6n}{5}\\ &= 7\cdot 2^k \cdot 4^0 + \frac{6}{5} + \frac{6n}{5} = \frac{35\cdot 2^k + 6 + 6n}{5}\end{aligned}$$

for following the edge $e_{\rightarrow}(v_0^0)$. Hence, agent i_1 follows the edge $e_{\rightarrow}(v_0^0)$.

For agent $i_x, x > 1$, at state v_0^0 the perceived cost is

$$\begin{aligned}B_{i_x}^T(0)\cdot c(e_{\nearrow}(v_0^0)) + d(v_1^1,\tau) &= b_{i_x} \cdot \Gamma_{i_x}^T(0) \cdot \frac{6}{5} + \frac{6n}{5}\\ &= 5\cdot 4^0 \cdot \frac{6}{5} + \frac{6n}{5} = \frac{30 + 6n}{5}\end{aligned}$$

for following edge $e_{\nearrow}(v_0^0)$ and

$$\begin{aligned}B_{i_x}^T(0)\cdot c(e_{\rightarrow}(v_0^0)) + d(v_0^1,\tau) &= b_{i_x} \cdot \Gamma_{i_x}^T(0) + \frac{6}{5} + \frac{6n}{5}\\ &= 5\cdot 4^0 + \frac{6}{5} + \frac{6n}{5} = \frac{31 + 6n}{5}\end{aligned}$$

for following the edge $e_{\rightarrow}(v_0^0)$. Hence, agent i_x chooses to follow edge $e_{\nearrow}(v_0^0)$.

At $t = 1$ agent i_1 is at state v_1^0 and all other agents are at state v_1^1.

For agent i_1 at state v_1^0 the perceived cost is

$$\begin{aligned}B_{i_1}^T(1)\cdot c(e_{\nearrow}(v_1^0)) + d(v_2^1,\tau) &= b_{i_1} \cdot \Gamma_{i_1}^T(1) \cdot \frac{6}{5} + \frac{6n}{5}\\ &= 7\cdot 2^k \cdot 4^{-(k-1)/2} \cdot \frac{6}{5} + \frac{6n}{5} = \frac{84 + 6n}{5}\end{aligned}$$

for following edge $e_{\nearrow}(v_1^0)$ and

$$\begin{aligned}B_{i_1}^T(1)\cdot c(e_{\rightarrow}(v_1^0)) + d(v_2^0,\tau) &= b_{i_1} \cdot \Gamma_{i_1}^T(1) + \frac{6}{5} + \frac{6n}{5}\\ &= 7\cdot 2^k \cdot 4^{-(k-1)/2} + \frac{6}{5} + \frac{6n}{5} = \frac{76 + 6n}{5}\end{aligned}$$

for following edge $e_{\nearrow}(v_1^0)$. Hence, agent i_1 follows edge $e_{\rightarrow}(v_1^0)$.

For agent $i_x, x > 1$, at state v_1^1 the perceived cost is

$$\begin{aligned}B_{i_x}^T(1)\cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2,\tau) &= b_{i_x} \cdot \Gamma_{i_x}^T(1) \cdot \frac{6n}{5}\\ &= 5\cdot 4^{1/2} \cdot \frac{6n}{5} = \frac{60n}{5}\end{aligned}$$

for following edge $e_{\nearrow}(v_1^1)$ and

$$\begin{aligned}B_{i_x}^T(1)\cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1,\tau) &= b_{i_x} \cdot \Gamma_{i_x}^T(1) \cdot n + \frac{6n}{5}\\ &= 5\cdot 4^{1/2} \cdot n + \frac{6n}{5} = \frac{56n}{5}\end{aligned}$$

for following edge $e_{\rightarrow}(v_1^1)$. Hence, agent i_x chooses to follow edge $e_{\rightarrow}(v_1^1)$.

Notice that after time $t = 2$, all agents continue procrastinating as their perceived cost does not change. Eventually, the progress path $p^T(i_1)$ of agent i_1 costs $\frac{6}{5} + n \cdot (n-2) + \frac{6n}{5}$ and the progress path $p^T(i_x)$ for an agent $i_x, x > 1$, costs $n - 2 + \frac{6}{5} + \frac{6n}{5}$. Hence, the total cost is $\Omega(k \cdot n^2)$.

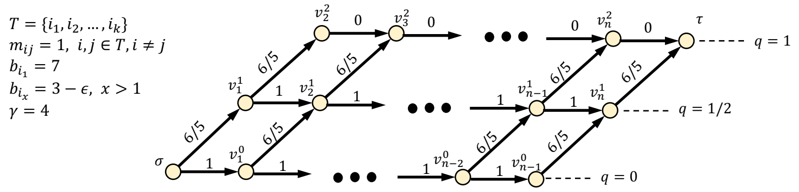


Figure 5 An example where collaboration leads to a decrease, by an $\Omega(n)$ factor, on the total cost.

Notice that in the case where $\Gamma^T_{i_z}(t) = 1$ for all $z \in T$, agent i_1 would still follow the same progress path, agent $i_x, x > 1$, would follow the edge $e_{\nearrow}(v_1^1)$ at state v_1^1 as its perceived cost would be:

$$B^T_{i_x}(1) \cdot c(e_{\nearrow}(v_1^1)) + d(v_2^2, \tau) = b_{i_x} \frac{6n}{5} = 5 \cdot \frac{6n}{5} = 6n$$

for following edge $e_{\nearrow}(v_1^1)$ and

$$B^T_{i_x}(1) \cdot c(e_{\rightarrow}(v_1^1)) + d(v_2^1, \tau) = b_{i_x}\ n\ +\ \frac{6n}{5} = 5 \cdot\ n + \frac{6n}{5} = \frac{31n}{5}$$

for following edge $e_{\rightarrow}(v_1^1)$. Hence, agent i_x would choose to follow edge $e_{\nearrow}(v_1^1)$. In this scenario, agent i_1 would pay cost $n - 2 + \frac{6}{5} + \frac{6n}{5}$ and $i_x, x > 1$, would pay cost $\frac{6}{5} + \frac{6n}{5}$. Hence, the total cost would be $\Omega(k \cdot n)$. Collectively, the total cost of progress paths can increase by a factor $\Omega(n)$. ◀

Proof of Lemma 8. Follows from Claim 5.1 from Gravin et al. [7] ◀

Proof of Lemma 9. See Figure 5. ◀

Proof of Lemma 10. In the case that $b_i = b_j$ the lemma is clearly true, assuming that i and j break ties consistently: the two agents will follow the exact same path. For the rest of the proof, and w.l.o.g., we assume that $b_i > b_j$. Note that for each t we have that $B_i^{\{i\}}(t) = b_i$ and $B_j^{\{j\}}(t) = b_j$. For the sake of leading to a contradiction assume that the statement of the lemma is false. Let v_t^ℓ be the first state that agent j went below agent i, formally, that $(v_t^\ell, v_{t+1}^{\ell+1}) \in p(i)$ and $(v_t^\ell, v_{t+1}^\ell) \in p(j)$. Let $p_i = \langle v_t^\ell = v_0, v_1^i, \dots, v_k^i = \tau \rangle$ be the subpath of $p(i)$ from v_t^ℓ to τ and $p_j = \langle v_t^\ell = v_0, v_1^j, \dots, v_k^j = \tau \rangle$ be the subpath of $p(j)$ from v_t^ℓ to τ. The perceived cost of the path p_i for i is $b_i \cdot c(v_0, v_1^i) + d(v_1^i, \tau)$. By the definition of $p(i)$ we have that

$$b_i \cdot c(v_0, v_1^i) + d(v_1^i, \tau) \leq b_i \cdot c(v_0, v_1^j) + d(v_1^j, \tau)$$

$$\Rightarrow b_i \cdot (c(v_0, v_1^i) - c(v_0, v_1^j)) + d(v_1^i, \tau) - d(v_1^j, \tau) \leq 0. \tag{1}$$

Similarly, for j we have that

$$b_j \cdot (c(v_0, v_1^j) - c(v_0, v_1^i)) + d(v_1^j, \tau) - d(v_1^i, \tau) \leq 0,$$

$$\Rightarrow b_j \cdot (c(v_0, v_1^j) - c(v_0, v_1^i)) - d(v_1^i, \tau) + d(v_1^j, \tau) \leq 0, \tag{2}$$

By the fact that $b_i > b_j$, and our assumption that when the perceived cost of making progress and procrastinating is equal then the agent chooses to make progress, it follows that at most one of the two inequalities can hold with equality. Summing the two inequalities, we obtain that

$$b_i \cdot (c(v_0, v_1^i) - c(v_0, v_1^j)) + b_j \cdot (c(v_0, v_1^j) - c(v_0, v_1^i)) < 0,$$

which, recalling that $v_0 = v_t^\ell$, that $v_1^i = v_{t+1}^{\ell+1}$, and that $v_1^j = v_{t+1}^\ell$, can be rewritten as

$$b_i \cdot (c(e_\nearrow(v_t^\ell)) - c(e_\rightarrow(v_t^\ell))) + b_j \cdot (c(e_\rightarrow(v_t^\ell)) - c(e_\nearrow(v_t^\ell))) < 0,$$

$$\Rightarrow (b_i - b_j) \cdot (c(e_\nearrow(v_t^\ell)) - c(e_\rightarrow(v_t^\ell))) < 0.$$

But this is a contradiction because we assumed that $b_i > b_j$ and by Property 6 it follows that $c(e_\nearrow(v_t^\ell)) \geq c(e_\rightarrow(v_t^\ell))$. ◀

Proof of Lemma 11. We prove it by contradiction. Let v_t^ℓ be the first node for which there exists an agent i for whom $(v_t^\ell, v_{t+1}^\ell) \in p^T(i)$ and $(v_t^\ell, v_{t+1}^{\ell+1}) \in p(i_1)$. Given that this is the first time that this happens, for each $j \in T$ we have that $q_j(t) \geq q_i(t)$. Note that by Property 2 we have that $\Gamma_i^T(t) \leq 1$, which implies $B_i^T(t) \leq \max\{b_i, 1\} \leq b_{i_1}$.

Arguing as in Lemma 10 we obtain that $(b_{i_1} - B_i^T(t))(c(e_\nearrow(v_t^\ell)) - c(e_\rightarrow(v_t^\ell))) < 0$, leading to a contradiction, which means that $p^T(i) \succeq_G p(i_1)$.

Repeating the argument, and observing that by Property 2 we have that $\Gamma_i^T(t) \geq 1$, giving $B_i^T(t) \geq \max\{b_i\, \Gamma_i^T(t), 1\} \geq b_{i_k}$, we obtain that $p^T(i) \preceq p(i_k)$. ◀

Proof of Lemma 13. First, we show that whenever, at some state v_t^ℓ agent i follows the edge $e_\nearrow(v_t^\ell)$, then i follows the shortest path from v_t^ℓ to σ. That is, $d(v_{t+1}^{\ell+1}, \tau) = d(v_t^\ell, \tau) - c(e_\nearrow(v_t^\ell))$. To prove our claim, we notice that

$$B_i^T(t) \cdot c(e_\nearrow(v_t^\ell)) + d(v_{t+1}^{\ell+1}, \tau) < B_i^T(t) \cdot c(e_\rightarrow(v_t^\ell)) + d(v_{t+1}^\ell, \tau).$$

$$\Rightarrow B_i^T(t) \cdot (c(e_\nearrow(v_t^\ell)) - c(e_\rightarrow(v_t^\ell))) < d(v_{t+1}^\ell, \tau) - d(v_{t+1}^{\ell+1}, \tau).$$

Since $B_i^T(t) \geq 1$ and $c(e_\nearrow(v_t^\ell)) > c(e_\rightarrow(v_t^\ell))$ by Property 6, we have

$$c(e_\nearrow(v_t^\ell)) - c(e_\rightarrow(v_t^\ell)) < d(v_{t+1}^\ell, \tau) - d(v_{t+1}^{\ell+1}, \tau),$$

which proves that $d(v_{t+1}^{\ell+1}, \tau) = d(v_t^\ell, \tau) - c(e_\nearrow(v_t^\ell))$. Hence, each time an agent follows an edge $e_\nearrow(v_t^\ell)$ from any state v_t^ℓ the remaining shortest path to τ decreases by $c(e_\nearrow(v_t^\ell))$ (by Property 5). Notice that this does not imply anything about the behavior of agent i at any other time $t' \neq t$. Our goal is to rely on Property 5 to guarantee that remaining shortest path cannot increase (independently of the followed path) and to bound the additional cost that each agent pays when not decreasing its distance to the target state. By our first claim, the progress path of agent i only increases (compared to the shortest path) when the agent procrastinates, that is, the agent follows an edge $e_\rightarrow(v_t^\ell)$ from some state v_t^ℓ.

Assume that i_{min} is the agent in T with the smallest personal bias $b_{i_{min}}$. Then, by Lemma 11, $p^T(j) \succeq p(i_{min})$, for all $j \in T, j \neq i_{min}$. Let $i \in T$ be any agent $i \neq i_{min}$. For any state v_t^ℓ such that i follows the edge $e_\rightarrow(v_t^\ell)$, agent i_{min} at time t at state $v_t^{\ell'}$ follows either the edge $e_\nearrow(v_t^{\ell'})$ or the edge $e_\rightarrow(v_t^{\ell'})$. By Properties 6 and 12 and the fact that $\ell' \leq \ell$ (since $p^T(i) \succeq p(i_{min})$) we have that $c(e_\nearrow(v_t^{\ell'})) \leq c(e_\rightarrow(v_t^{\ell'})) \leq c(e_\rightarrow(v_t^\ell))$. That is, each time agent i follows an edge $e_\rightarrow(v_t^\ell)$ from some state v_t^ℓ, agent i_{min} follows an edge with larger cost when operating independently. As the total increase in the progress path of agent i, compared to the shortest progress path, is bounded by the number of procrastination edges

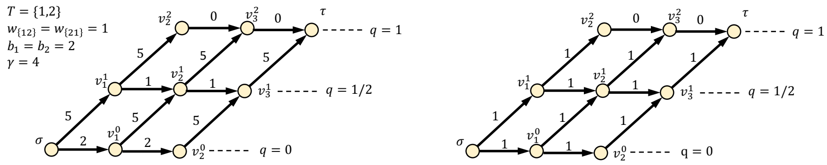


Figure 6 Graphs of type 1 (left) and type 2 (right) that are used in the proof of Theorem 16.

followed by agent i, it follows that the total increase is bounded by the cost of the progress path $p(i_{min})$. Hence, each agent adds at most cost equal to the cost of the progress path $p(i_{min})$. That is, the total cost increases by a factor of at most k. ◀

Proof of Theorem 16. We obtain a reduction from the SetCover problem. In the SetCover problem, we are given a universe of items U and a family of sets $C_1, C_2, \ldots, C_\ell$ and we are asked to find k sets such that each element from the universe is contained in at least one selected set. Our overall strategy is to construct an instance of the OptimalGroupAssignment problem, in polynomial time, from an instance of the SetCover problem. This means, if we can solve the OptimalGroupAssignment problem in polynomial time, then our reduction is a polynomial time algorithm for the SetCover problem, which is known to be **NP**-hard.

Assume that we are given an instance of the SetCover problem with universe U and sets $C_1, C_2, \ldots, C_\ell$. We construct an instance of the OptimalGroupAssignment problem as follows. We include k graphs of type 2 and $\ell + |U| - k$ graphs of type 1, as they are shown in Figure 6. For each element $u \in U$, we include an *element-agent* a_u. For each set $C_i \in \{C_1, C_2, \ldots, C_\ell\}$, we include a *set-agent* a_i. All agents have personal bias $b = 2$, and $\gamma = 2$. Finally, we set $w_{ui} = 1$ if $u \in C_i$, where $C_i \in \{C_1, C_2, \ldots, C_\ell\}$ and $u \in U$, and $w_{ui} = 0$ otherwise. Notice that any agent operating on a type-2 graph always pays cost 2 as it always follows the path $\langle s, v_1^1, v_2^1, v_3^2, t\rangle$. Therefore, the total cost only depends on the agents that operate on type 1 graphs. An agent operating individually on a type-1 graph follows the path $\langle s, v_1^0, v_2^0, v_3^1, t\rangle$, and therefore, pays 14. If for an agent i, that is operating on a type-1 graph, it holds that $w_{ji} = 1$ and agent j is operating on a type-2 graph, agent i follows the path $\langle s, v_1^0, v_2^1, v_3^1, t\rangle$ with cost 13. In the case where $w_{ji} = 0$, for all agents $j \neq i$, or $w_{ji} = 1$ but agent j follows a path containing v_1^0 (i.e., operates on a type 1 graph), it holds that i follows the path $\langle s, v_1^0, v_2^0, v_3^1, t\rangle$ with cost 14.

We show that for any instance of the SetCover problem there exist k sets covering all elements of the universe if and only if there is a solution to the OptimalGroupAssignment problem such that the total cost is $k \cdot 2 + |U| \cdot 13 + (\ell - k)14$. We begin with the first direction, that is, we show that if there exists a solution to the SetCover problem, then there exists an assignment in the OptimalGroupAssignment problem for which the total cost is $k \cdot 2 + |U| \cdot 13 + (\ell - k)14$. For each set C_i in the solution of the SetCover problem, we assign its corresponding set-agent a_i to a type-2 graph (they pay collectively $k \cdot 2$ cost). For the rest set-agents corresponding to sets C_j of the SetCover instance, it holds $w_{ij} = 0$, for all $i \neq j$, and therefore those $(\ell - k)$ agents collectively pay $(\ell - k) \cdot 14$ cost (i.e., they all follow the path $\langle s, v_1^0, v_2^0, v_3^1, t\rangle$ as we explained above). Finally, each element-agent u pays cost $|U| \cdot 13$ in total, as there exists a weight $w_{iu} = 1$, for some $i \neq u$ such that a_i operates on a type 2 graph (as there is a set C_i covering u). This proves the first direction.

Now we prove that if there exists an assignment in the OptimalGroupAssignment problem for which the total cost is $k \cdot 2 + |U| \cdot 13 + (\ell - k)14$ then there is a solution to the SetCover problem. Notice that, independently of which agents operate on graphs of type

2, they always pay cost 2. Every agent a_u, for $u \in U$, pays cost 2 if it operates on a type-2 graph, cost 13 if it operates on a type-1 graph and there exists $w_{iu} = 1$ where a_i operates on a type-2 graph, and 14 otherwise. Note that for each $w_{iu} = 1$, there exists a set C_i covering the element u in the SetCover instance Therefore, the only agents that can pay cost 13 are the element-agents. Since in any solution there are exactly k agents paying cost 2 (i.e., those operating on type-2 graphs) and at least $(\ell - k)$ agents paying cost 14 (i.e., the set-agents that operate on type-2 graphs), the minimum possible total cost is $k \cdot 2 + |U| \cdot 13 + (\ell - k)14$, which is achieved by assigning k set-agents to type-2 graphs, such that for each element-agent u it holds $w_{iu} = 1$ for at least one of the set-agents that were assigned to type-2 graphs. Such an assignment indicates that there exists a set of k set-agents $a_1, a_2, \ldots, a_k$ such that for each $u \in C$, there exists a $1 \leq i \leq k$ such that $w_{iu} = 1$. These k set-agents correspond to a solution of the SetCover instance. This concludes the proof. ◀

Walking Through Doors Is Hard, Even Without Staircases: Proving PSPACE-Hardness via Planar Assemblies of Door Gadgets

Joshua Ani
Massachusetts Institute of Technology, Cambridge, MA, USA
joshuaa@mit.edu

Jeffrey Bosboom
Massachusetts Institute of Technology, Cambridge, MA, USA
jbosboom@mit.edu

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA
edemaine@mit.edu

Yenhenii Diomidov
Massachusetts Institute of Technology, Cambridge, MA, USA
diomidov@mit.edu

Dylan Hendrickson
Massachusetts Institute of Technology, Cambridge, MA, USA
dylanhen@mit.edu

Jayson Lynch
Massachusetts Institute of Technology, Cambridge, MA, USA
jaysonl@mit.edu

Abstract

A door gadget has two states and three tunnels that can be traversed by an agent (player, robot, etc.): the “open” and “close” tunnel sets the gadget’s state to open and closed, respectively, while the “traverse” tunnel can be traversed if and only if the door is in the open state. We prove that it is PSPACE-complete to decide whether an agent can move from one location to another through a *planar* assembly of such door gadgets, removing the traditional need for crossover gadgets and thereby simplifying past PSPACE-hardness proofs of Lemmings and Nintendo games Super Mario Bros., Legend of Zelda, and Donkey Kong Country. Our result holds in all but one of the possible local planar embedding of the open, close, and traverse tunnels within a door gadget; in the one remaining case, we prove NP-hardness.

We also introduce and analyze a simpler type of door gadget, called the *self-closing* door. This gadget has two states and only two tunnels, similar to the “open” and “traverse” tunnels of doors, except that traversing the traverse tunnel also closes the door. In a variant called the *symmetric* self-closing door, the “open” tunnel can be traversed if and only if the door is closed. We prove that it is PSPACE-complete to decide whether an agent can move from one location to another through a *planar* assembly of either type of self-closing door. Then we apply this framework to prove new PSPACE-hardness results for several 3D Mario games and Sokobond.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases gadgets, motion planning, hardness of games

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.3

Related Version `https://arxiv.org/abs/2006.01256`

Acknowledgements This work was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) taught by Erik Demaine in Spring 2019. We thank the other participants of that class for related discussions and providing an inspiring atmosphere.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 3; pp. 3:1–3:23
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Puzzle video games are rife with doors that block the player's passage when closed/locked. To open such a door, the player often needs to collect the right key or keycard, or to press the right combination of buttons or pressure plates, or to solve some other puzzle. Many of these game features in sufficient generality imply that the video game is NP-hard or PSPACE-hard, according to a series of metatheorems starting at FUN 2010 [7, 9, 10].

An intriguing twist is to use doors as a framework for proving hardness of video games that do not "naturally" have doors, but have some mechanics that suffice to simulate doors via a gadget. The first use of a local "door gadget" was by Viglietta to prove Lemmings PSPACE-complete at FUN 2014 [11]. This door gadget is a portion of a level design containing three directed paths that the player can traverse: a "traverse" path that can be traversed if and only if the door is open, a "close" path that forces the door to close, and an "open" path that allows the player to open the door if desired. Viglietta [11, Metatheorem 3] proved that such a door gadget, together with the ability to wire together door entrance/exit ports according to an arbitrary graph (including crossovers for a 2D game like Lemmings), where the player has the choice of how to traverse the graph, suffice to prove PSPACE-hardness of deciding whether an agent can move from one location to another. At the same FUN, Aloupis et al. [1] used this door framework to prove Legend of Zelda: Link to the Past and Donkey Kong Country 1, 2, and 3 PSPACE-complete [1]. At the next FUN, Demaine et al. [6] used this door framework to prove Super Mario Bros. PSPACE-complete. All of these proofs feature a crossover gadget for wiring paths between door gadgets.

The motion-planning-through-gadgets framework of [4, 5] (initiated at FUN 2018) formalizes the idea of moving one or more agents through a graph of local gadgets, where each gadget has local state and traversal paths whose traversal affects that gadget's state (only). In the 1-player unbounded setting considered here, past work analyzed gadgets that are

1. ***deterministic***, meaning that when an agent enters a gadget at any location, it has a unique exit location and causes a unique state change;
2. ***reversible***, meaning that every such traversal can be immediately undone, both in terms of agent location and gadget state change; and
3. ***k-tunnel***, meaning that the $2k$ entrance/exit locations can be paired up such that, in any state, traversal paths only connected paired locations (in some direction).

Restricted to deterministic reversible k-tunnel gadgets, Demaine et al. [5] characterized which gadget sets make motion planning of an agent from one location to another PSPACE-complete: whenever the gadget set contains a gadget with ***interacting tunnels***, meaning that traversing some traversal path changes (adds or removes) the traversability of some other traversal path (in some direction). Furthermore, they proved the same characterization when the gadgets are connected in a planar graph, obviating the need for a crossover gadget.

Door gadgets naturally fit into this motion-planning-through-gadgets framework. (Indeed, they were one of the inspirations for the framework.) Notably, however, the door gadget used in [1, 6, 11] is neither deterministic (the open path can open the door or not, according to the player's choice) nor reversible (the paths are all directed in fixed directions), so the existing characterization and planarity result do not apply.

In this paper, we develop a specialized motion-planning-through-*doors* framework, completing another subspace of the motion-planning-through-gadgets framework. Our framework applies to a variety of different door gadgets, including the door gadget of [1, 6, 11]. In all cases, a door gadget has two states and three disjoint traversal paths: "traverse", "close", and "open". Each path may be individually ***directed*** (traversable in one direction) or ***undirected*** (traversable in both directions). In addition, the open traversal path may have identical entrance and exit locations, meaning that its traversal changes the door's state but does not move the agent (breaking the k-tunnel assumption). In this way, we can require that traversing the open and close traversal paths force the door's state to open and closed, respectively, but still effectively allow the player to make a choice of whether to open the door (by skipping or including the open traversal, which leaves the agent in the same location either way).

In Section 2, we introduce two more families of door gadgets. A ***self-closing door*** has two states but only two traversal paths: "open" and "self-close". The self-close traversal is possible only in the open state, and it forcibly changes the state to closed. As before, each traversal path can be either directed or undirected; and the open traversal forces the state to open, but we allow the open traversal path to have identical start and end locations, which effectively allows optional opening. A ***symmetric self-closing door*** has two states and two traversal paths: "self-open" and "self-close". The self-open/close traversal is possibly only in the closed/open state, respectively, and it forcibly changes the state to open/closed, respectively. (This definition is fully symmetric between "open" and "close".) Each traversal path can be either directed or undirected, but we no longer allow optional traversal.

In Section 3, we prove that *planar* 1-player motion planning is PSPACE-complete for every door gadget, for every local combinatorial planar embedding of every type door gadget except for one (which we only prove NP-hard). Thus, all that is needed to prove a new game PSPACE-hard is to construct any single supported door gadget, and to show how to connect the door entrances/exits together in a planar graph. In particular, the crossover gadgets previously constructed for Lemmings [11, Figure 2(e)], Legend of Zelda: Link to the Past and Donkey Kong Country 1, 2, and 3 [1, Figures 28 and 20], and Super Mario Bros. [6, Figure 5] are no longer necessary for those PSPACE-hardness proofs: they can now be omitted. (See Section 4 for details.) Our result should therefore make it easier in the future to prove 2D games PSPACE-hard. Because of their reduced conceptual complexity – only two traversal paths, which behave essentially identically for symmetric self-closing doors – we have found it even easier to prove games PSPACE-hard by building self-closing door gadgets.

In the full version of the paper we prove that every door is ***universal***, meaning that any one of them can simulate *all* gadgets in the motion-planning-through-gadgets framework of [4, 5]. This provides the first examples of fully universal gadgets.

In Section 4, we illustrate this approach by proving PSPACE-hardness for one 2D game, Sokobond, and several different 3D Mario games: Super Mario 64, Super Mario 64 DS, Super Mario Sunshine, Super Mario Galaxy, and Captain Toad: Treasure Tracker (and the associated levels in Super Mario 3D World). Additional applications to Super Mario Galaxy 2 and Super Mario 3D Land/World are presented in the full version of the paper. These reductions consist of just one gadget, a symmetric self-closing door, along with easy methods for connecting these gadgets. For the 3D games, the main benefit is the simplicity of the symmetric self-closing door: crossovers are generally easy in the 3D games, though conveniently we still do not need to explicitly build them.

2 Self-Closing Doors

In this section, we introduce different kinds of self-closing doors and show that 1-player motion planning is PSPACE-hard for them.

2.1 Terminology

A ***self-closing door*** is a 2-state gadget that has a tunnel that closes itself when traversed (the *self-closing* tunnel), a tunnel/port that reopens said tunnel (the *opening* tunnel/port), and no other ports. We will talk about two major kinds of self-closing door. A ***normal self-closing door*** is a self-closing door where the open path/tunnel is always open. A ***symmetric self-closing door*** is a self-closing door where the open path/tunnel is a tunnel and also closes itself when traversed. As with doors, these can be ***directed***, ***undirected***, or ***mixed***, and a normal self-closing door can also be ***open-required*** or ***open-optional***. An 'X' on a tunnel indicates that the tunnel closes itself when traversed. A dotted line indicates a closed tunnel and a solid line indicates an open tunnel. For normal self-closing doors, the open path/tunnel will be colored green. Figure 1 shows some self-closing doors.

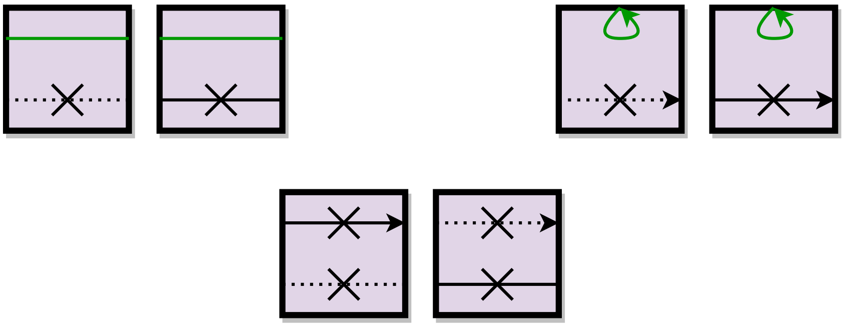


Figure 1 Left: An undirected open-required normal self-closing door. Right: A directed open-optional normal self-closing door. Bottom: A mixed symmetric self-closing door.

2.2 PSPACE-hardness of Self-Closing Doors

In this section we show PSPACE-hardness for 1-player motion planning with any of the self-closing doors. We do so by showing undirected self-closing doors can simulate diodes, and self-closing doors without open-optional tunnels can simulate ones with open-optional tunnels. We then prove the main Theorem 2.3 which gives PSPACE-hardness of the directed, open-optional, normal self-closing door by simulating a directed, open-optional door gadget.

▶ **Lemma 2.1.** *In 1-player motion planning, any normal or symmetric self-closing door can simulate an open-optional self-closing door.*

Proof. In the case of an open-optional normal self-closing door, we are done. In the case of an open-required normal self-closing door, we do the same thing we did for the proof for Theorem 3.6. In the case of a symmetric self-closing door, we pick a tunnel to be the opening tunnel and do what we did for Theorem 3.6. This simulates an open-optional self-closing door. ◀

▶ **Lemma 2.2.** *1-player motion planning with the undirected open-optional normal self-closing door can simulate a directed open-optional normal self-closing door.*

Proof. We can simulate a diode by wiring 2 undirected open-optional normal self-closing doors as shown in Figure 4. The player can enter from the left, open the left self-closing door, traverse it, and do the same for the right self-closing door. The player cannot enter from the right. If the player tries to open the left self-closing door and then leave, the player still cannot enter from the right. If the player tries to open the right self-closing door and then leave, they will not be able to leave. So this simulates a diode. We can wire a diode to each side the self-closing tunnel to get a directed self-closing tunnel which can be applied to make the undirected self-closing door directed. ◀

▶ **Theorem 2.3.** *1-player motion planning with the directed open-optional normal self-closing door is PSPACE-hard.*

Proof. We can simulate a diode by wiring the opening port to the input end of the self-closing tunnel. The player can open the self-closing tunnel then traverse it, but cannot go the other way because the self-closing tunnel is directed. Then we show that we can duplicate the open port and the self-closing tunnel as in Figure 2. We then actually triplicate the open port and duplicate the self-closing tunnel, and wire them up to simulate the directed open-optional door as shown in Figure 3, for which PSPACE-hardness is known. ◀

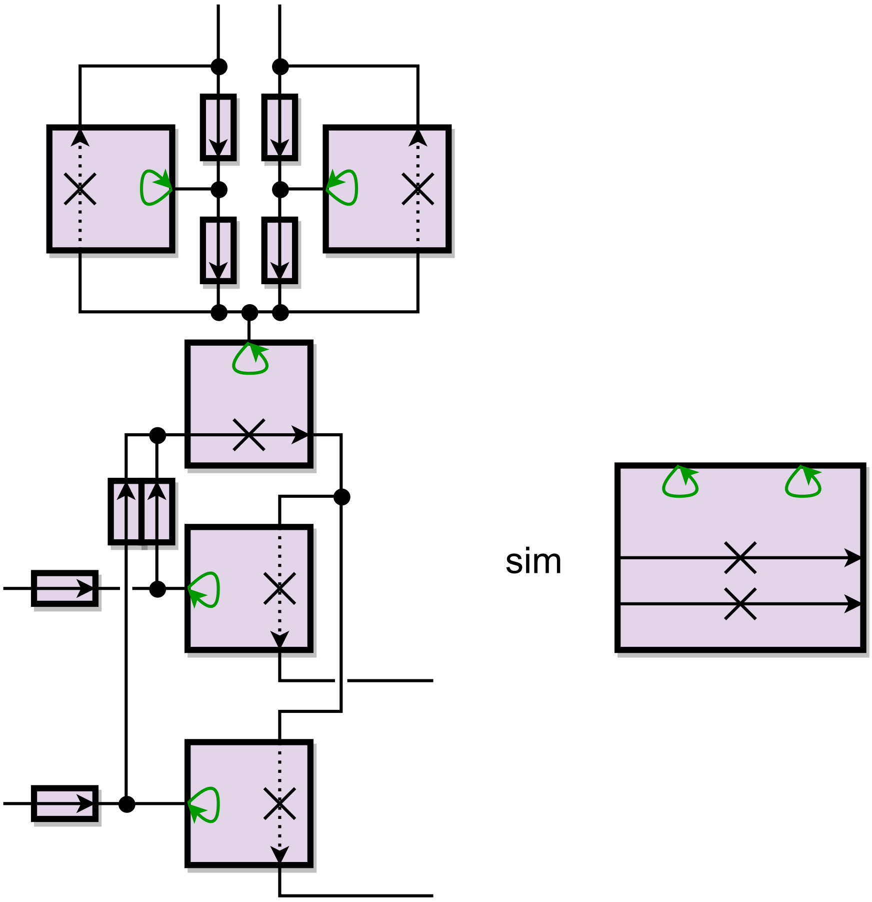


Figure 2 The directed open-optional normal self-closing door can simulate a version of itself with the opening port and the self-closing tunnel duplicated. Note that the opening port duplicator is planar.

Chaining the simulations in Lemmas 2.1 and 2.2 with Theorem 2.3 we obtain PSPACE-hardness for all variations.

▶ **Corollary 2.4.** *1-player motion planning with any normal, symmetric, or open-optional normal self-closing door is PSPACE-hard.*

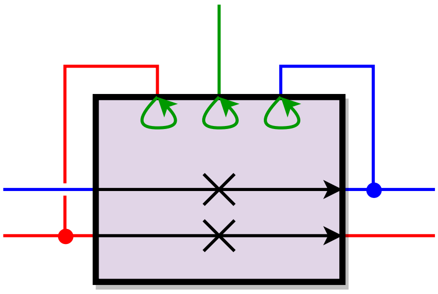


Figure 3 Simulation of the directed open-optional door. Green wires correspond to the opening port; blue wires correspond to the traverse tunnel; and red wires correspond to the closing tunnel. Note that the player has no reason to not open the gadget after traversing the blue wire.

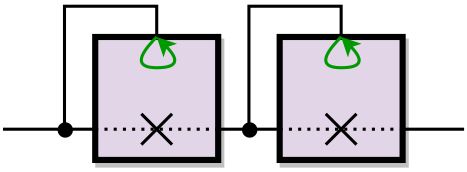


Figure 4 Undirected open-optional normal self-closing door simulating a diode.

3 Planar Doors

In this section, we adapt the door framework of [1, Section 2.2] (a cleaner presentation of the framework from [11]) into the motion-planning-through-gadgets framework. We then improve upon those results by showing most variations on the door gadget remain PSPACE-hard in the planar case. We also show that 1-player planar motion planning with any normal or symmetric self-closing door is PSPACE-hard.

3.1 Terminology

We define a ***door*** to be a gadget with an *opening* port or tunnel, a *traverse* tunnel, and a *closing* tunnel, and each of the tunnels may be directed or undirected. The opening port/tunnel opens the traverse tunnel, and the closing tunnel closes the traverse tunnel. Throughout this paper, the opening port/tunnel will be colored green, the traverse tunnel will be colored blue, and the closing tunnel will be colored red. In addition, a solid traverse tunnel represents an open door, and a dotted traverse tunnel represents a closed door. A ***directed door*** is a door where all tunnels are directed. An ***undirected door*** is a door where all tunnels are undirected. A door that is neither undirected nor directed is a ***mixed door***. An ***open-required door*** is a door with an opening tunnel, and an ***open-optional door*** is one with an opening port. A directed open-required door, an undirected open-required door, and a mixed open-optional door are shown in Figure 5.

In 2D, we care about the arrangement of ports in a gadget. For *planar motion planning* problems we want a *planar* system of gadgets, where the gadgets and connections are drawn in the plane without crossings. Planar gadgets also specify a clockwise ordering of their ports, although we consider rotations and reflections of a gadget to be the same. A single gadget type thus corresponds to multiple planar gadget types, depending on the choice of the order of locations. For a planar system of gadgets, the gadgets are drawn as small diagrams with points on their exterior coorisponding to their ports and connections are drawn as paths connecting the points corresponding to the ports without crossing gadget interiors or other connections.

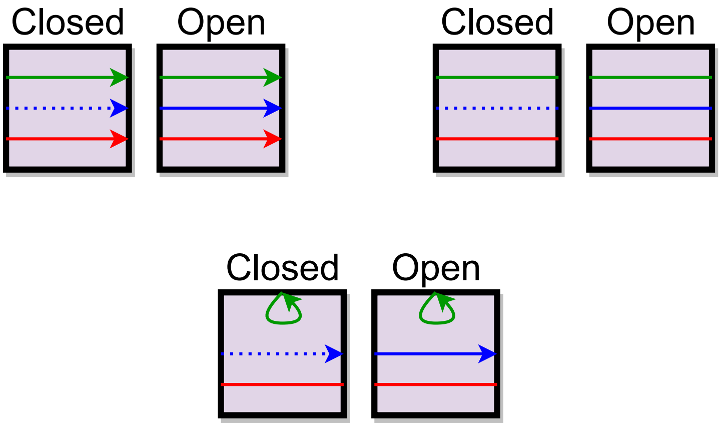


Figure 5 Left: A directed open-required door. Right: An undirected open-required door. Bottom: A mixed open-optional door.

3.2 PSPACE-hardness for Planar Self-Closing Doors

For completeness, we give a proof that the planar directed open-optional normal self-closing door is PSPACE-hard. This result was also given in [2].

▶ **Theorem 3.1.** *1-player planar motion planning with the directed open-optional normal self-closing door is PSPACE-hard.*

Proof. Since Theorem 2.3 shows PSPACE-completeness in the non-planar case, it will suffice to build a crossover gadget. First, we wish to duplicate the opening ports as in the prior proof. We show how to do so in Figure 2. Note that this time we cannot directly duplicate the self-closing tunnel as the construction from Theorem 2.3 uses crossovers. We can also simulate a diode as proven in Theorem 2.3 since the construction is planar. We use these to simulate a pair of self-closing doors where the opening ports alternate which door they open, shown in Figure 6. If the agent enters from port 1 or 4, they will open door E or F, respectively, and then leave. If the agent enters from port 2, they can open doors A, B, and C. Assume they then traverse door B. If they then open door E, they would have to traverse door C, maybe open F, and get stuck. So instead of opening door E, the agent traverses door A, ending up back at port 2 with no change except that door C is open. Entering port 2 or 3 gives the opportunity to open door C without being forced to take a different path, so leaving door C open does not help. So instead of traversing door B, the agent traverses door C. The agent is then forced to go right and can open door F. Then they are forced to traverse door B. If the agent opens door E, they will be stuck, so the agent traverses door A instead and returns to port 2, leaving door F open. Similarly, if the agent enters from port 3, the only useful thing they can do is open port E and return to port 3.

Using this, we then simulate a directed crossover as in Figure 7 which are able to simulate an undirected crossover, removing the planar restriction and reducing this problem to Theorem 2.3. In the simulation of a directed crossover, the agent must open the left tunnel of a gadget and then open both tunnels of the other one, forcing them to cross over, since the only path forward goes through the left tunnels of both gadgets. ◀

▶ **Theorem 3.2.** *1-player planar motion planning with any normal or symmetric self-closing door is PSPACE-hard.*

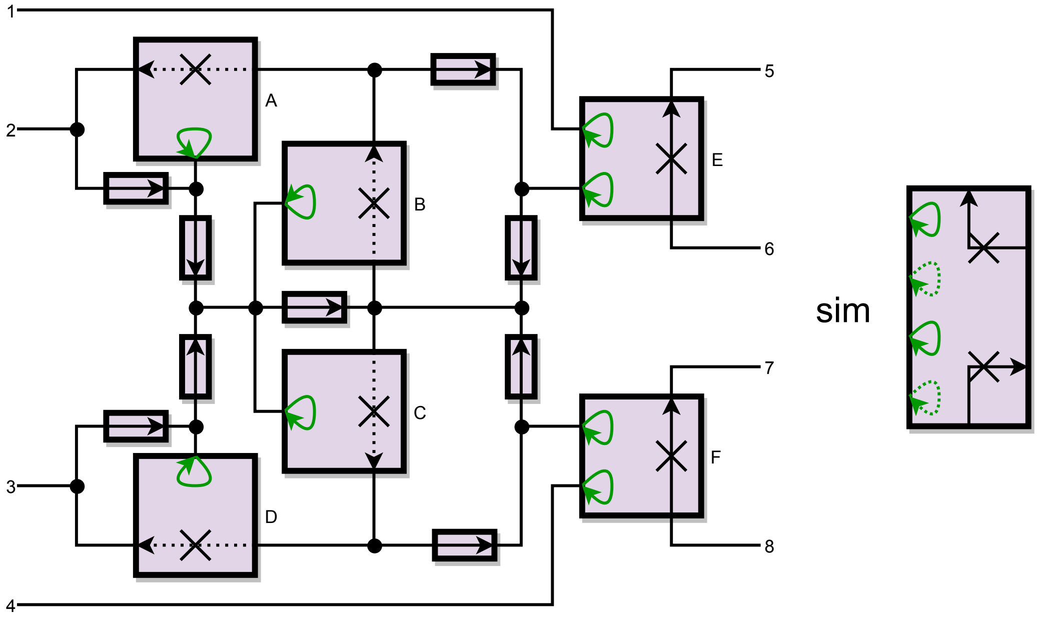


Figure 6 Directed open-optional normal self-closing door simulating the gadget on the right, where solid opening ports control the top self-closing tunnel and dotted opening ports control the bottom self-closing tunnel. The gadgets and external ports are labelled to help with the proof.

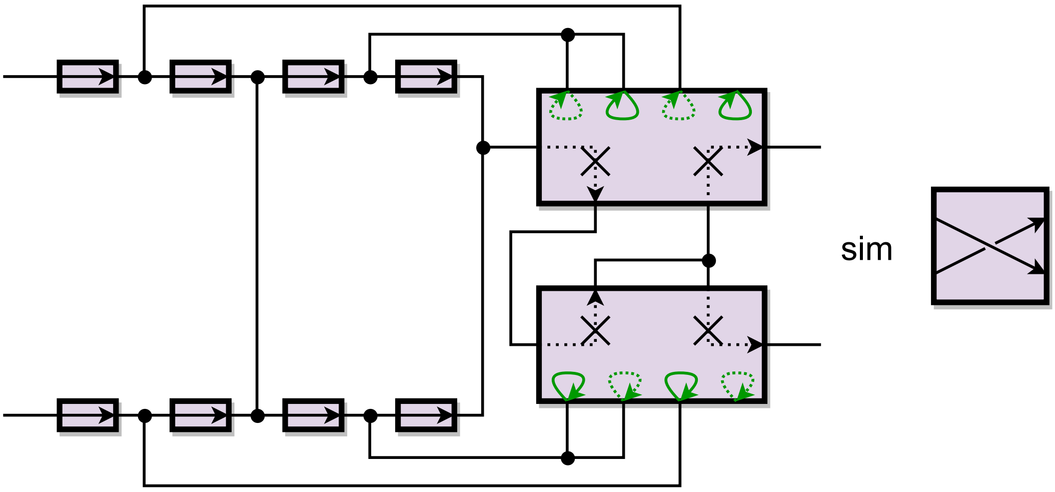


Figure 7 Directed open-optional normal self-closing door simulating a crossover.

Proof. Any normal or symmetric self-closing door can simulate a diode as shown in Figure 8(a–f). Then we can simulate the directed open-optional normal self-closing door as shown in Figure 9(a–d). Finally we apply Theorem 3.1 to show PSPACE-hardness. ◀

3.3 PSPACE-hardness for Planar Doors

We will show that 1-player planar motion planning with almost any door is PSPACE-hard by showing that 1-player planar motion planning with almost any fully directed door is PSPACE-hard and that mixed and undirected doors can planarly simulate at least one of the PSPACE-hard fully directed doors.

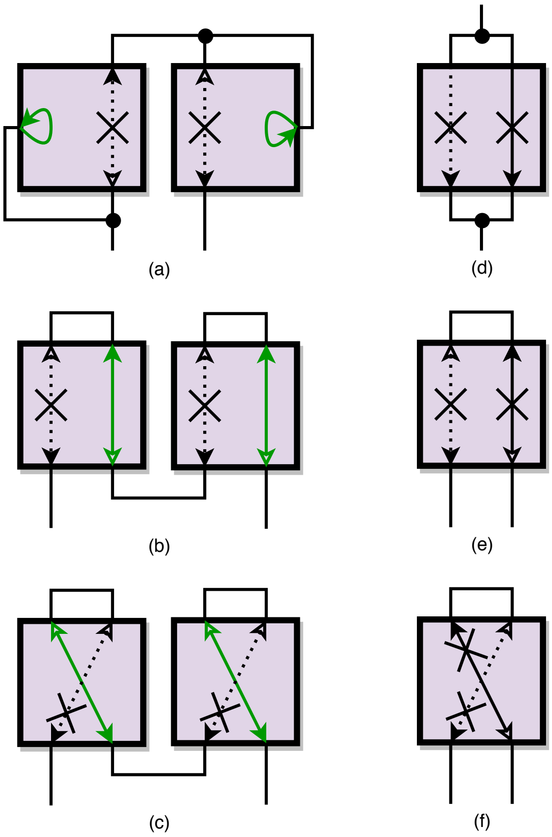


Figure 8 Six types of self-closing doors simulating diodes. Filled-in arrows indicate directions that are required to exist, and outlined arrows indicate optional directions. Case (a) is the same as Figure 4.

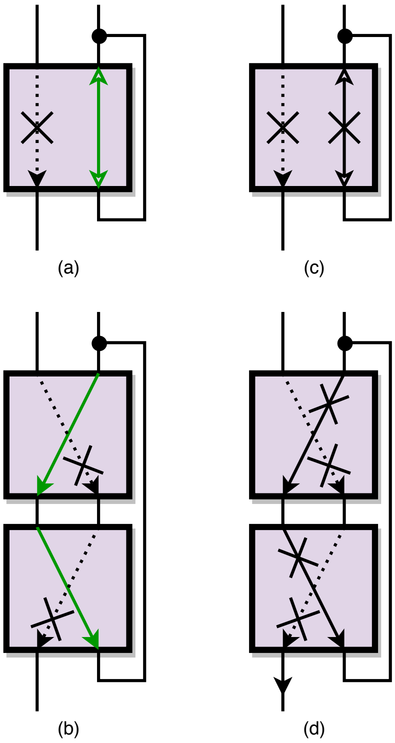


Figure 9 Four types of directed self-closing doors simulating the directed open-optional normal self-closing door. Filled-in arrows indicate directions that are required to exist, and exactly one of the outlined directions must exist.

We first show that mixed and undirected doors can simulate fully directed doors in Lemmas 3.3 and 3.4. Since undirected and partially directed doors can planarly simulate at least one fully directed door, it suffices to prove hardness for all fully directed doors. Next, we show hardness for all fully directed doors with at least one pair of crossing tunnels. We then show we can collapse adjacent opening ports to optional opening ports in Theorem 3.6, this leaves 12 fully directed doors with no crossing tunnels (Figure 10)s. These 12 doors are shown and named in Figure 10. Proofs for 11 of the 12 of these cases are given in Theorem 3.8. Finally, we show NP-hardness for the remaining Case 8: OTtocC door in Theorem 3.11.

▶ **Lemma 3.3.** *Any mixed door can planarly simulate some fully directed door which is not the Case 8: OTtocC door.*

Proof. Consider an arbitrary mixed door M. Since M is mixed, it has a directed tunnel. No tunnel changes its own traversability when crossed, so this tunnel simulates a diode. We wire each undirected tunnel of M through diodes at each end pointing in the same direction. This simulates a directed door. If M is not the door in Case 8: OTtocC, we are done. Otherwise, flip one set of diodes wired through an undirected tunnel of M, simulating a different directed door. ◀

▶ **Lemma 3.4.** *An undirected door can planarly simulate a fully directed door which is not the Case 8: OTtocC door.*

Proof. Consider an arbitrary undirected door U. We wire an external wire to a port of the opening port/tunnel. The player can visit the port, or if it is a tunnel, cross the tunnel both ways, to open the gadget. If the opening port/tunnel was a tunnel, this turns it into a port, making the gadget U'. Consider the order of the ports of the opening port O, the traverse tunnel $\{T_0, T_1\}$, and the closing tunnel $\{C_0, C_1\}$ around the edge of U', and label the ports p_0, p_1, p_2, p_3, p_4. We want to show that a traverse tunnel port is adjacent to a closing tunnel port. Assume not. Without loss of generality, let $p_0 = T_0$. Then $\{p_1, p_4\} = \{T_1, O\}$. But then $\{p_2, p_3\} = \{C_0, C_1\}$, and one of $\{p_2, p_3\}$ must be adjacent to a traverse tunnel port, a contradiction. Since one of the traverse tunnel ports, say T_1, is adjacent to one of the closing tunnel ports, say C_0, we wire T_1 to C_0 without blocking an opening port or opening tunnel port. This simulates a directed open-optional normal self-closing door: The player can open the gadget by going to the opening port (or if it is a tunnel, by going through the tunnel and back). If the gadget is open, the player can go through the traverse tunnel and then the closing tunnel, but cannot go the other way. If the gadget is closed, the player cannot go either way through the traverse-tunnel-closing-tunnel path. ◀

▶ **Theorem 3.5.** *1-player planar motion planning with any directed door with an internal crossing is PSPACE-hard.*

Proof. If the opening tunnel crosses the closing tunnel, then we have a crossover because these tunnels are always open. If the opening tunnel crosses the traverse tunnel, then we start the door open and have a crossover because neither tunnel closes itself or the other. Otherwise, the traverse tunnel crosses the closing tunnel and the opening port/tunnel can

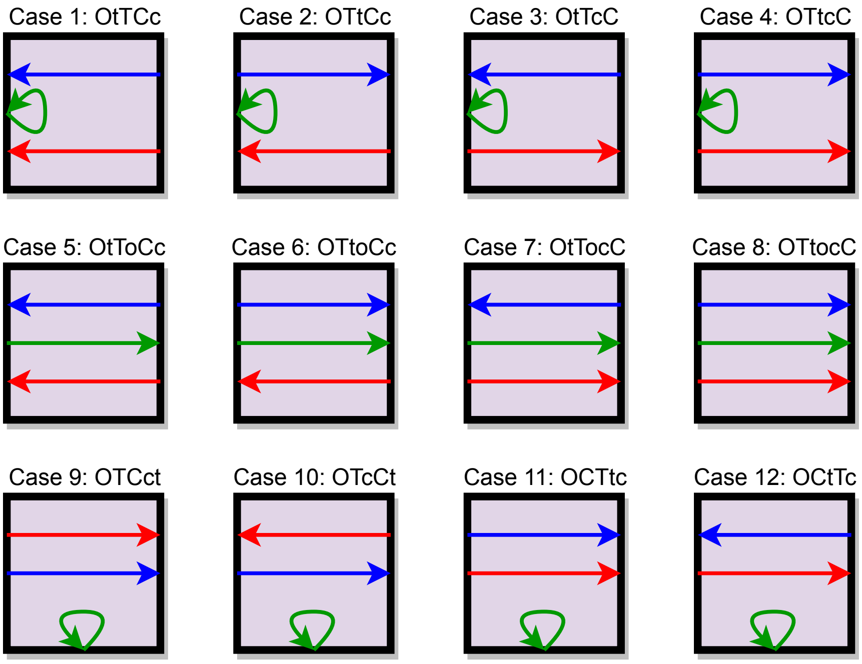


Figure 10 The twelve cases of a planar directed door without internal crossings. Opening tunnels with adjacent ports are merged into opening ports.

simulate an opening port. Then we have four cases, as shown in Figure 11. In cases 1, 2, and 4, we can simulate a crossover by connecting the opening port to either the input of the traverse tunnel or the output of the closing tunnel to ensure that the traverse tunnel is open when we need to use it. (Figure 12).

Case 3, however, is more tricky, as both of these ports are separated from the opening port by other ports. We use 2 copies to provide a path from the input of the traverse tunnel to the opening port without giving access to the close tunnel. The horizontal path of the crossover involves crossing from the left door to the right door, which is allowed as long as the left door is open. To take the vertical path, the player opens the middle door, goes down closing the left door, opens the right door, traverses the middle door, opens the left door (to keep the horizontal path open), and traverses the right door. The player can leave partway through this traversal, but this does nothing useful. So all doors with internal crossings can simulate crossovers, removing the planarity constraint. ◀

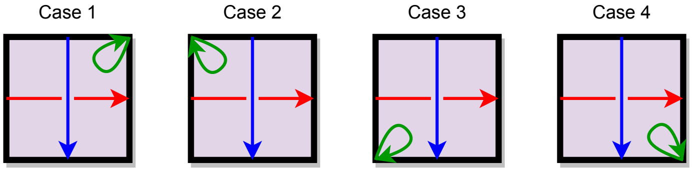


Figure 11 The four cases where the traverse tunnel crosses the closing tunnel but the opening port/tunnel does not cross either and can thus simulate a port.

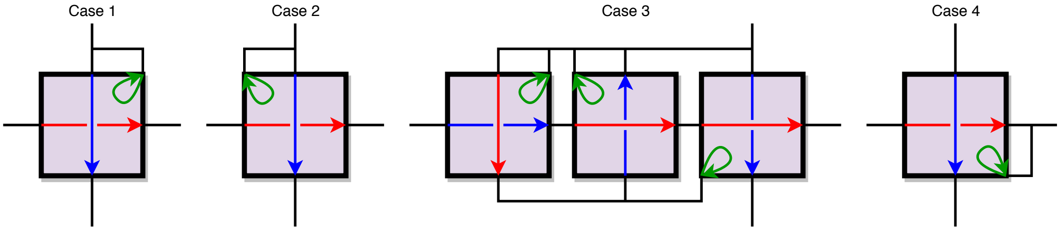


Figure 12 All four cases of the traverse tunnel crossing the closing tunnel can each simulate a crossover.

▶ **Theorem 3.6.** *In 1-player motion planning, any door can simulate its corresponding open-optional door.*

Proof. In case of a door that is not already open-optional, we wire one end of the open tunnel to the other end and wire some point on this loop externally as shown in Figure 13. This turns the open tunnel into an open port. ◀

Before continuing, we prove another gadget, the directed tripwire lock, is PSPACE-complete. Recall that a tripwire lock is a 2-state 2-tunnel gadget with an undirected tunnel that is traversable in exactly 1 state and an undirected tunnel that toggles the state of the gadget [4]. The ***directed tripwire lock*** is similar except that its tunnels are directed.

▶ **Lemma 3.7.** *1-player planar motion planning with the parallel directed tripwire-lock is PSPACE-hard.*

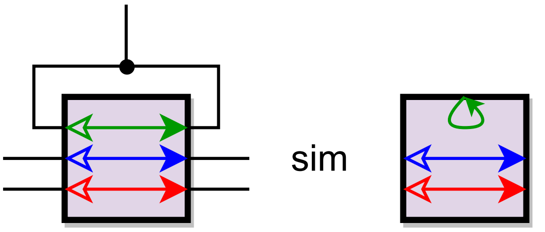


Figure 13 An open-required door simulates its corresponding open-optional door. Outlined arrows indicate optionally allowed traversals.

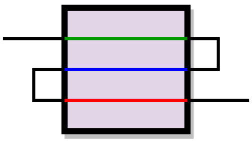


Figure 14 Simulation of a diode with an undirected door.

A proof can be found in the full version of the paper.

For directed doors, there are only the cases without internal crossings left. If the opening port/tunnel is a tunnel and its ports are adjacent, we easily simulate an opening port, reducing the number of cases to consider. There are twelve cases, shown in Figure 10. We name these cases based on the cyclic order of ports, with exits-only having lowercase letters.

▶ **Theorem 3.8.** *1-player planar motion planning with any directed door without internal crossings except the Case 8: OTtocC door is PSPACE-hard.*

Proof. We divide into multiple cases. Note the cases are numbered according to Figure 10, not in the order they are addressed in this proof.

Case 2: OTtCc, Case 10: OTcCt, and Case 12: OCtTc doors. In all these doors the opening port/tunnel is a port, and the traverse tunnel output is adjacent to the closing tunnel input. Thus, we can simulate a directed open-optional self-closing door by wiring the traverse tunnel output to the closing tunnel input and by attaching a wire to the open port, and these wires do not cross each other. Then this reduces to Theorem 3.2.

Case 1: OtTCc door. can simulate the directed version of the tripwire lock, as shown in Figure 15. We will refer to the gadgets numbered left to right. The lock is simply the traverse tunnel on door 1. In the two simulated states we will either have doors 1 and 3 open or door 2 open. If door 2 is open, when traversing the tripwire tunnel we can go through the traverse tunnel allowing us to open doors 1 and 4. With door 4 now open, we can go through its traverse tunnel opening door 3, and then closing door 4 on the way out. This leaves us with doors 1 and 3 open. Going through the tripwire tunnel again closes door 1 but allows us to go through the traverse tunnel of door 3, allowing us to open door 2. Doors 3 and 4 are then closed on the way out. There are states where we could fail to open all of these doors while traversing the close tunnel, but this will leave the gadget with strictly less traversability and thus the agent will never want to take such a path. Thus the Case 1: OtTCc door is PSPACE-complete by Lemma 3.7.

Case 3: OtTcC door. This door can simulate a directed open-optional normal self-closing door (Figure 16). If the agent enters from port O (the opening port), they can open doors 2 and 3. If they then leave, they have accomplished nothing because door 2 was already open, and door 3 can be opened from port O anyway and cannot be traversed from port T_0 or T_1 as we will see later. So they close door 2 instead. Then they can open door 1 and they are forced to traverse door 3. The agent can then reopen door 2 and return to port O. Now all the doors are open. If the agent then enters from port T_0, then they are forced to close door

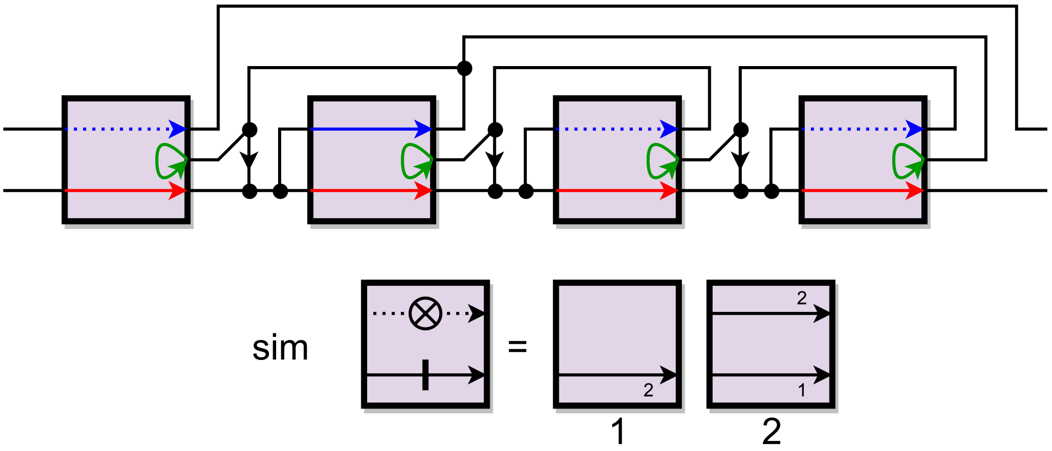


Figure 15 The Case 1: OtTCc door simulates the parallel directed tripwire lock. In addition, the state diagram of the directed tripwire lock. Arrows are drawn directly on wires to represent diodes.

3. They can then open door 1 (useless), and then they are forced to traverse door 2 and close door 1, leading to port T_1. The agent could not have taken this path initially because door 1 was closed, and they cannot take it again without visiting port O because they just closed door 1.

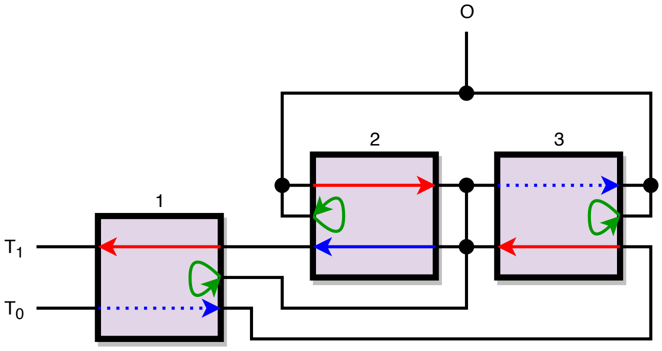


Figure 16 Simulation of a self-closing door with the Case 3: OtTcC door. The simulation starts in the closed state. Ports and gadgets are labelled.

Case 4: OTtcC door. A proof of this case can be found in the full paper.

Case 6: OTtoCc door. This door can simulate a directed open-optional normal self-closing door (Figure 17). If the agent enters from port O, they are forced to close door 3. If the agent then traverses door 2, they are forced to open door 3 and return to port O, accomplishing nothing. So the agent has no other option but to close door 1. If the agent tries to open door 2, they get stuck, so they instead open door 1. Continuing the loop involving door 1 does nothing, so the agent then traverses door 2, opens door 3, and returns to port O. Now door 1 is open. If the agent enters from port T_0, then they are forced to close door 2, traverse door 1, and close door 1. Reopening door 1 puts the agent back into the situation of being forced to close door 1, so the agent instead opens door 2 and traverses door 3 to port T_1.

The agent could not have taken this path initially since door 1 was closed, and they cannot take it again without visiting port O because they closed door 1.

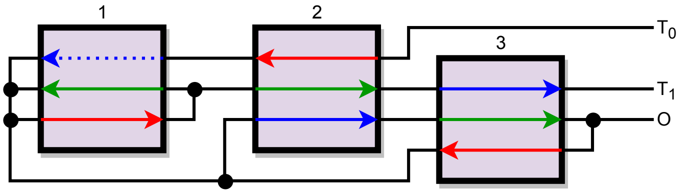


Figure 17 Simulation of a self-closing door with the Case 6: OTtoCc door. The simulation starts in the closed state. Ports and gadgets are labelled.

Case 5: OtToCc door. This door can simulate the Case 6: OTtoCc door, which has been covered, by effectively flipping the traverse tunnel. (Figure 18). Door 1 is the gadget that we flip the traverse tunnel of. If the agent enters from port T_0, they must open door 2, the close door 2. If door 1 is open and the agent then traverses it, they are back to a previous position with nothing changed. Instead, the agent opens door 3. If the agent then closes door 3, they get stuck because door 2 is closed. So they must close door 2 (again) or traverse door 3. These actions lead to the same situation. If the agent opens door 3 (again), they are back to the same situation that occurred after opening door 3 the first time. If door 1 is open, the agent then traverses door 1. Then they must open door 2. Closing door 2 leads to a previous situation, so the agent then traverses door 3. If the agent then traverses door 1 (again), they must open door 2 (again), leading to a previous situation. So they instead open door 3. Closing door 2 and traversing door 3 lead to different previous situations, so the agent then closes door 3, and then is forced to traverse door 2 to port T_1, leaving all the doors unchanged. If door 1 is not open, then the agent is unable to leave.

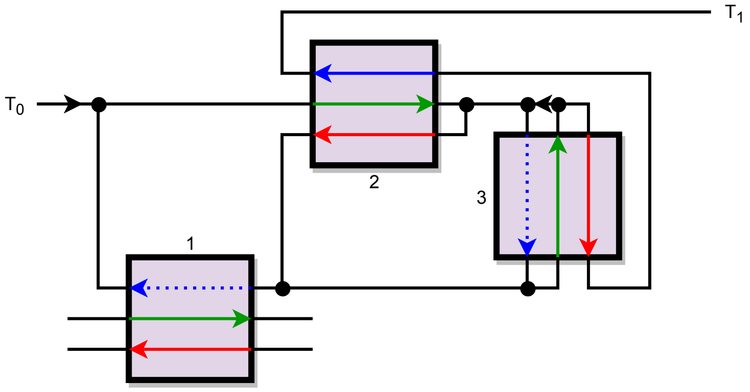


Figure 18 Simulation of the Case 6: OTtoCc with the Case 5: OtToCc door. The traverse tunnel of the leftmost gadget is effectively flipped.

Case 7: OtTocC door. This door can simulate a directed open-optional normal self-closing door (Figure 19). If the agent enters from port O, they must open door 1, then close door 2. If the agent then closes door 3, they get stuck because door 2 is closed. The agent can traverse door 1 and leave via port O, but they can also open and then traverse door 3 and then do the same thing, which is advantageous. So the agent opens and traverses door 3, then traverses door 1 to port O. Now door 1 is open, door 2 is closed, and door 3 is open. If the agent enters from port T_0, they must close door 1, then open door 2, then traverse door 3. Opening door 3 and then traversing it is a no-op, and door 1 is closed, so the agent closes door 3 and then must traverse door 2 to port T_1. This leaves door 1 closed, door 2 open, and door 3 closed. The agent could not have taken this path initially because door 3 was closed, and cannot take it again without visiting port O first for the same reason.

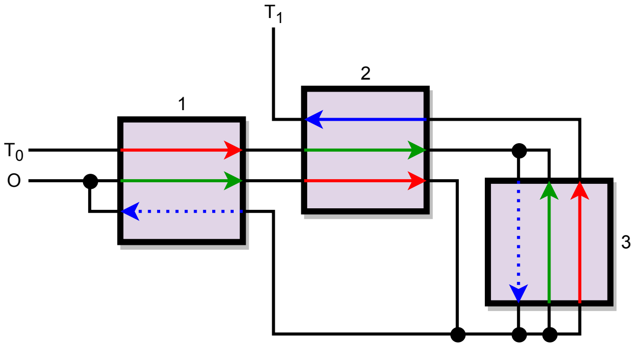


Figure 19 Simulation of a self-closing door with the Case 7: OtTocC door.

Case 9: OTCct door. This door can simulate a directed open-optional normal self-closing door (Figure 20). If the agent enters from port O, they can open door 1 and must close door 2. If the agent later enters from port T_0, then they must traverse door 1. They then can open door 2 (and must, since that is the only way out) and must close door 1. Then the agent traverses door 2 to port T_1. The agent could not have taken this path initially because door 1 was closed, and cannot take the path again without visiting port O first for the same reason.

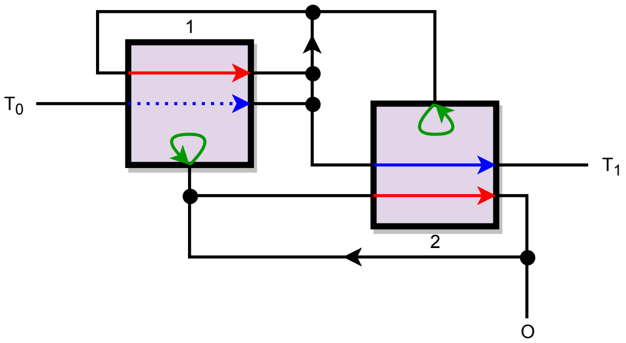


Figure 20 Simulation of a self-closing door with the Case 9: OTCct door.

Case 11: OCTtc door. A proof of this case can be found in the full paper.

This covers all the planar directed doors without internal crossings except the OTtocC door, finishing the proof. ◀

▶ **Theorem 3.9.** *1-player planar motion planning with any door except the door in Case 8: OTtocC is PSPACE-hard.*

Proof. This follows from Theorems 3.5, 3.8, 3.3, and 3.4, as those cover all the cases. ◀

To prove NP-hardness of the last case (Case 8: OTtocC), we first prove NP-hardness of other useful gadgets. A ***NAND*** gadget is a directed 2-tunnel gadget where traversing either tunnel closes both tunnels (preventing all future traversals). There are three planar types of NAND gadgets, named by analogy with 2-toggles [4]: one ***crossing*** type (where the two tunnels cross); and two noncrossing types, ***parallel*** (where the directions are the same) and ***antiparallel*** (where the directions are opposite). The notion of NAND gadgets was introduced in [3], which proved NP-hardness using a combination of parallel and antiparallel NAND gadgets, "one-way" gadgets, "fork" gadgets, and "XOR" gadgets. We prove that NAND gadgets alone suffice:

▶ **Lemma 3.10.** *1-player planar motion planning is NP-hard with either antiparallel NAND gadgets or crossing NAND gadgets.*

Proof. Figures 21 and 22 show that antiparallel NAND gadgets can simulate crossing NAND gadgets and vice versa. Figure 23 shows how crossing NAND gadgets can simulate parallel NAND gadgets. Therefore we can assume the availability of all three planar types of NAND gadgets.

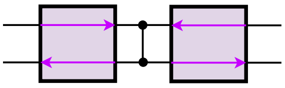


Figure 21 Simulation of crossing NAND gadget by antiparallel NAND gadgets.

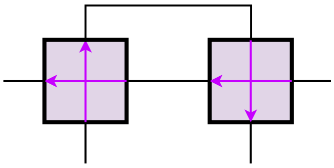


Figure 22 Simulation of antiparallel NAND gadget by crossing NAND gadgets.

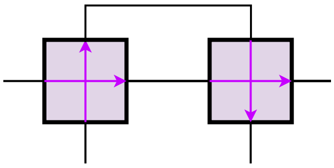


Figure 23 Simulation of parallel NAND gadget by crossing NAND gadgets.

We follow the NP-hardness reduction from Planar 3-Coloring to Push-1-X in [3]. This reduction requires four types of gadgets. Their "NAND gadget" is our parallel and antiparallel (noncrossing) NAND gadgets, which we have. Their "XOR-crossing gadget" is a crossing 2-tunnel gadget that breaks down (in a particular way) if both tunnels get traversed. The reduction guarantees that at most one tunnel in an XOR-crossing gadget will be traversed (because they correspond to different color assignments), so we can replace this gadget with a crossing NAND gadget (which even prevents both tunnels from being traversed). Their "fork gadget" is a one-entrance two-exit gadget such that either traversal closes the other traversal; we can simulate this gadget with a parallel NAND gadget by connecting together the two entrances. Their "one-way gadget" is a gadget that prevents traversal in one direction, but provides no constraint after being traversed in the other direction. Because this gadget is required only to block certain traversals, and each gadget gets visited only once (in particular because the reduction is to Push-1-X where the robot is not permitted to revisit a square), we can replace this gadget with a NAND gadget where one tunnel is not connected to anything. Therefore we have established NP-hardness using only NAND gadgets. ◀

▶ **Theorem 3.11.** *1-player planar motion planning with the door in Case 8: OTtocC is NP-hard.*

Proof. We show how to simulate antiparallel NAND gadgets, which is NP-hard by Lemma 3.10. First, Figure 24 shows how to combine two Case 8: OTtocC doors to build a door-like gadget with an open tunnel and two traverse–close tunnels, where traversing the open tunnel opens both traverse–close tunnels, and traversing either traverse–close tunnel closes the other traverse–close tunnel. Next, Figure 25 shows how to combine two of these gadgets to build an antiparallel NAND gadget. The top tunnel in the top gadget is initially closed, forcing the agent to open it and thus close the bottom tunnel of the bottom gadget, which is possibly only if the bottom tunnel of the bottom gadget was not already traversed. Because the open tunnel of the bottom gadget is not connected to anything, both tunnels of the bottom gadget will remain closed once closed. ◀

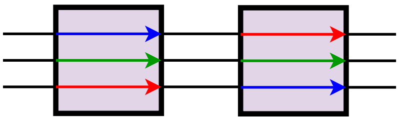


Figure 24 Simulation of parallel double-close door with the Case 8: OTtocC door.

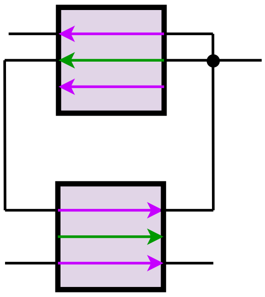


Figure 25 Simulation of an antiparallel NAND gadget with a parallel double-close door.

4 Applications

In this section we use our results about the complexity of door gadgets to prove PSPACE-hardness for seven new video games: Sokobond, and several different 3D Mario games. More applications are in the full paper.

Sokobond is a 2D block pushing game where the blocks are able to fuse into polyominoes. The Mario games considered are all 3D platformers in which the player controls Mario in an attempt to collect resources or reach target locations while avoiding or defeating enemies and environmental hazards. The player's main actions are having Mario jump and walk in an approximately continuous environment. Mario also has health and ways of taking damage which can cause the player to lose the game. More details on the needed additional mechanics are given in the section for each game. Captain Toad: Treasure Tracker is a 3D puzzle platformer and is mechanically similar to Mario except that Toad is unable to jump.

In addition, our planar door results simplify prior uses of a door framework. The Lemmings door [11, Figure 4] has an internal crossing, so Theorem 3.5 applies. The Donkey Kong Country 1, 2, and 3 doors [1, Figures 21–23] are the Case 10: OTcCt door, Case 4: OTtcC door, and internal crossing door, respectively, so Theorems 3.8 and 3.5 applies. The Legend of Zelda: A Link to the Past door [1, Figure 30] has an internal crossing, so Theorem 3.5 applies. The Super Mario Bros. door [6, Figure 6] is is the Case 4: OTtcC door, so Theorem 3.8 applies. Therefore all of the crossover gadgets in these reductions [11, Figure 2(e)], [1, Figure 20], [1, Figure 28], [6, Figure 5] are not in fact needed to prove PSPACE-hardness of these games.

4.1 Sokobond

Sokobond [8] is a 2D block pushing game where the blocks are atoms/molecules. Movement is discrete along a square grid. The player starts as a single atom. Each atom except He has some number of free electrons (H has 1, O has 2, N has 3, C has 4). When two atoms that both have free electrons are adjacent, they both lose a free electron and bond into a molecule. Molecules are rigid, so pushing an atom in a molecule results in the entire molecule moving. Atoms/molecules can also push each other.

Sokobond with He atoms is trivially NP-hard as it includes Push-$*$ [3]. We show PSPACE-hardness even without He atoms:

▶ **Theorem 4.1.** *Completing a level in Sokobond with H and O atoms is PSPACE-hard.*

Proof. We reduce from 1-player planar motion planning with a door that is not the Case 8: OTtocC door and use Theorem 3.9.

Let the player start as an H atom trying to reach another H atom. We can simulate a door that is not the Case 8: OTtocC door as shown in Figure 26. To open the door, the player pulls down on the big molecule. The player can go through the traverse tunnel if and only if the molecule is down. When going through the closing tunnel, the player is forced to push up on the molecule, closing the traverse tunnel. The molecule used to simulate a door has no free electrons, so the level can be completed if and only if the player can reach the other H atom. ◀

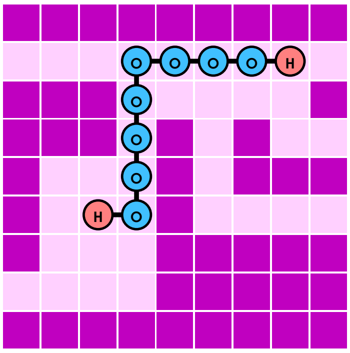


Figure 26 Simulation of a door in Sokobond. The opening port is at the bottom left. The traverse tunnel is undirected and runs between the top left and the top right. The closing tunnel is undirected and runs between the middle right and the bottom right.

4.2 Captain Toad: Treasure Tracker

Captain Toad: Treasure Tracker is a 3D puzzle platformer in the Mario universe, originally appearing as a type of level in Super Mario 3D World, and then released as a stand-alone game on the Wii U and ported to the 3DS and Switch. Notably, Toad can fall but not jump. The game contains rotating platforms controlled by a wheel which Toad must be adjacent to to move. The platforms move in 90° increments. We show PSPACE-hardness by constructing an antiparallel symmetric self-closing door (Theorem 2.4).

▶ **Theorem 4.2.** *Collecting Stars in a Captain Toad: Treasure Tracker is PSPACE-hard assuming no level size limit.*

Proof. Figure 27 gives a top-down view of the construction. There is a U-shaped rotating platform at a height slightly below the high ground and far above the low ground. The U-shaped platform rotates counterclockwise and can be reached from the nearby high ground; however, the gap between the back of the U and the other side is too far for Toad to jump. Further, the dividing wall sits slightly above the rotating platform, preventing Toad from crossing. Toad is able to go onto the U platform from the high ground, activate the gear twice, and jump off of the U platform onto the low ground across the gap. The U platform is now facing the other way, allowing Toad to enter from the high ground on the other side, but preventing other traversals. ◀

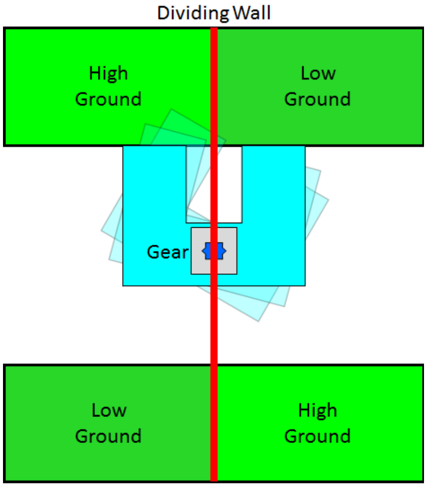


Figure 27 Top view of a simulation of a symmetric self-closing door.

4.3 Super Mario 64/Super Mario 64 DS

Super Mario 64 is a 3D Mario game for the Nintendo 64 where Mario collects Stars from courses inside paintings to save the princess, who is trapped behind a painting. Super Mario 64 DS is a remake of Super Mario 64 for the Nintendo DS (still in 3D), featuring the same courses as in Super Mario 64 plus new courses, as well as the ability to play as characters other than Mario. In this reduction, we will primarily make use of quicksand, which will defeat Mario if he lands in it, and the ghost enemy Boo.

The Boo is an enemy that (with normal parameters) chases Mario if he is looking away from it and is less than a certain distance away. Once Mario gets too far, the Boo moves back to its original position. Unlike most enemies, jumping on a Boo does not kill it, but instead sends it a short distance forward or backward, which we will use to help Mario cross the quicksand. Some walls stop the Boo but it can go through certain walls that normal Mario cannot go through, we call these Boo-only walls. The Boo is also unable to go through doors. We also make use of one-way walls which Mario and the Boo can go through in one direction but not the other.

For the setup, we use one Boo in Super Mario 64 DS and two Boos in Super Mario 64. Performing a kick while in the air sends Mario a short distance up and can normally only be performed once per jump. But Mario can kick after jumping on a Boo in Super Mario 64 DS even if he already kicked, allowing him to jump on the same Boo. This is not true in Super Mario 64, so jumping on a second Boo is necessary to stall long enough to jump on the first Boo again.

▶ **Theorem 4.3.** *Collecting a Star in a Super Mario 64/Super Mario 64 DS course is PSPACE-hard assuming no course size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to reach is a Star. The simulation is shown in Figure 28.

In the setup below, Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by going through the door on the bottom-left and hopping on the Boo(s) to the top-left. Then Mario lets the Boo(s) chase him a little to turn the Boo(s), and hops on the Boo(s) to push it into the top-right. Finally, Mario goes through the top-left door. Mario cannot just jump to the other side because the distance is too far. He also cannot go into the traverse path because of the Boo-only wall. The Boo(s) will try to go back to its home, but cannot because it is stuck behind a 1-way wall and a regular wall. If Mario does not move the Boo(s) to the top-right, it still cannot get back to its home because of a different 1-way wall, so Mario cannot leave the port 1 to port 2 traversal open.

Mario goes from port 3 to port 4 by going through the top-right door and hopping on the Boo(s) to the bottom-right, then going through the bottom-right door. The Boo(s) will go back to its original position at the bottom left on its own.

Mario cannot lure the Boo(s) away from the gadget because it is completely walled in except for the doors, which the Boo(s) cannot go through. ◀

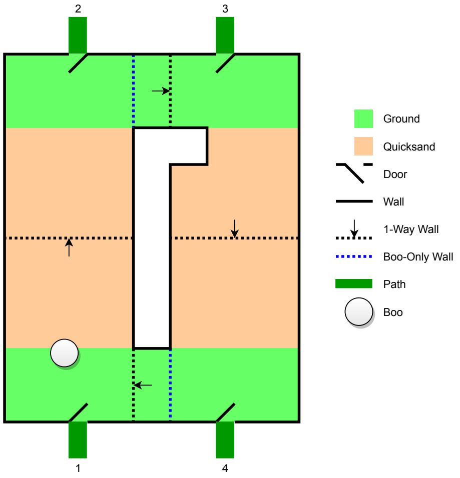


Figure 28 Simulation of a symmetric self-closing door in Super Mario 64 DS. In Super Mario 64, there are 2 Boos instead of 1. The ground and quicksand are on the same vertical level. The room is covered by a ceiling. The hallways are too wide to wall jump across.

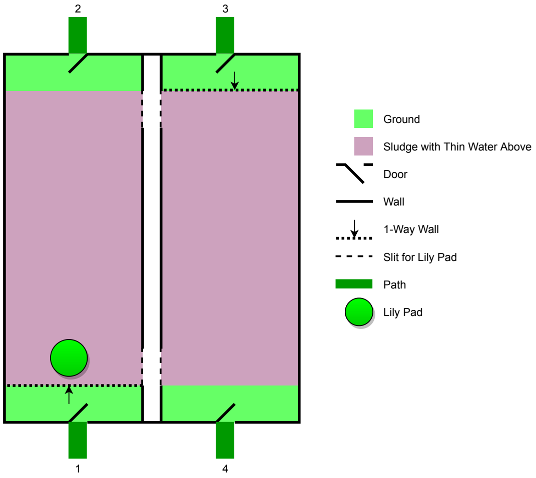


Figure 29 Simulation of a symmetric self-closing door in Super Mario Sunshine. The slits allow the Lily Pad to cross without allowing bulky Mario to do so. The hallways are too wide to wall jump across.

4.4 Super Mario Sunshine

Super Mario Sunshine is a 3D Mario game for the GameCube where Mario is falsely accused of spreading graffiti and is forced to clean it up before he can leave. Like Super Mario 64, this game includes one-way walls. This game features a new device, F.L.U.D.D., attached to Mario's back that allows him to spray water. Lily Pads float on water; the player can ride a Lily Pad and cause it to move by spraying water in the opposite direction. Sludge is an environmental hazard which kills Mario if he touches it. The general goal of a level is to collect Shrine Sprites.

▶ **Theorem 4.4.** *Collecting a Shine Sprite in a Super Mario Sunshine level is PSPACE-hard assuming no level size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to collect is a Shine Sprite. The simulation of a symmetric self-closing door is shown in Figure 29.

The thin water above the sludge prevents the Lily Pad from disintegrating, while preventing Mario from crossing without using the Lily Pad. Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by crossing the 1-way wall and riding the Lily Pad across, then moves the Lily Pad partially across the slit so it can be accessed from the other side. He cannot leak to the section between port 3 and port 4 because the slits are too thin. The sludge is too long to simply jump to the other side, so the Lily Pad is needed. Mario cannot do anything from port 2 because the 1-way wall blocks him from going to port 1. Mario goes from port 3 to port 4 in a similar manner. ◀

4.5 Super Mario Galaxy

Super Mario Galaxy is a 3D Mario game for the Wii where Mario goes to space. He encounters alien creatures along the way and collects Power Stars to restore the power of a spaceship. The game features downward gravity, upward gravity, sideways gravity, spherical gravity, cubical gravity, tubular gravity, cylindrical gravity that allows infinite freefall, W-shaped gravity, gravity that cannot make up its mind, and most importantly, controllable gravity.

Dark matter disintegrates Mario when he touches it, resulting in death. The Gravity Switch changes the direction of gravity when spun and can be spun multiple times.

▶ **Theorem 4.5.** *Collecting a Power Star in a Super Mario Galaxy galaxy is PSPACE-hard assuming no galaxy size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to collect is a Star. The simulation of a symmetric self-closing door is shown in Figure 30.

The Gravity Switch in this construction switches gravity between down and up. Mario goes from port 1 to port 2 by crossing the 1-way wall and hitting the Gravity Switch on his way to the right. This is forced because of a pit of dark matter, and closes the port 1 to port 2 traversal because when gravity points up, attempting the traversal would land Mario on dark matter. At the same time, it opens the port 3 to port 4 traversal. Mario cannot enter port 2 and do anything useful because flipping the Gravity Switch means falling in the pit of dark matter. Mario goes from port 3 to port 4 in a similar manner. ◀

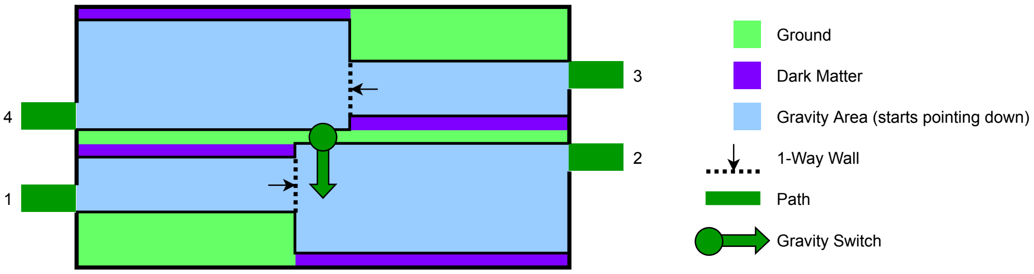


Figure 30 Simulation of a symmetric self-closing door in Super Mario Galaxy. This is a side view and is essentially 2-dimensional.

4.6 Super Mario Odyssey

Super Mario Odyssey is a 3D Mario game for the Switch where Mario travels to different kingdoms collecting Power Moons and eventually goes to the Moon. Mario has the ability (via his hat Cappy) to capture certain enemies and objects to use their powers, but such objects tend to reset position after being uncaptured, so we will not be using them here.

We make use of a Jaxi, poison, and timed platforms. A Jaxi is a statue lion that can be ridden safely across poison, which is a hazard that kills Mario. A timed switch makes some event happen for a specific amount of time. In our reduction, timed switch X makes platform X appear for just long enough for Mario to make a traversal.

▶ **Theorem 4.6.** *Collecting a Power Moon in a Super Mario Odyssey kingdom is PSPACE-hard assuming no kingdom size limit.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to reach is a Power Moon. The simulation of a symmetric self-closing door is shown in Figure 31.

Mario goes from port 1 to port 2 by pressing timed switch A, riding the Jaxi to the right, and traversing platform A. This opens the port 3 to port 4 traversal while closing the port 1 to port 2 traversal. Mario cannot go to port 3 because of the wide gap, or to port 4 because platform B is gone. The Jaxi is required because the poison it is on is very wide. Mario cannot do anything useful if he tries to enter from port 2 or port 4 because the platforms would be gone. Mario goes from port 3 to port 4 in a similar manner. ◀

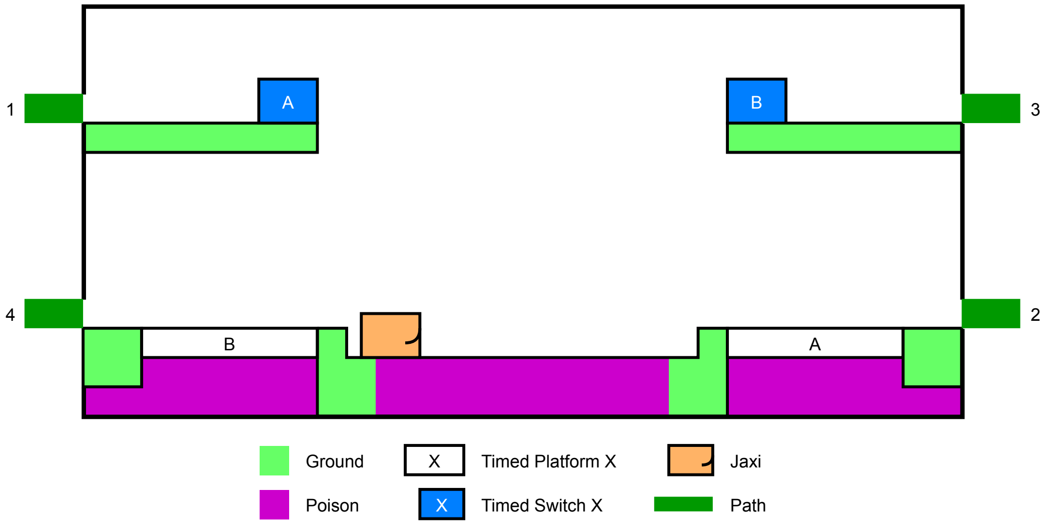


Figure 31 Simulation of a symmetric self-closing door in Super Mario Odyssey. This is a side view and is essentially 2-dimensional. All strips of poison are way too wide for Mario to cross with his various aerial skills, and the platforms with timed switches are too high to get to from below.

References

1 Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015. Originally appeared at FUN 2014.

2 Joshua Ani, Sualeh Asif, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. PSPACE-completeness of pulling blocks to reach a goal. In *Abstracts from the 22nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2019)*, pages 31–32, Tokyo, Japan, September 2019.

3 Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O'Rourke. Pushing blocks is hard. *Computational Geometry: Theory and Applications*, 26(1):21–36, August 2003.

4 Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 18:1–18:21, La Maddalena, Italy, June 2018.

5 Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 62:1–62:42, Seattle, January 2020. doi:10.4230/LIPIcs.ITCS.2020.62.

6 Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is harder/easier than we thought. In *Proceedings of the 8th International Conference on Fun with Algorithms (FUN 2016)*, pages 13:1–13:14, La Maddalena, Italy, June 2016.

7 Michal Forišek. Computational complexity of two-dimensional platform games. In *Proceedings of the 5th International Conference on Fun with Algorithms (FUN 2010)*, volume 6099 of *Lecture Notes in Computer Science*, 2010. doi:10.1007/978-3-642-13122-6_22.

8 Alan Hazelden, Lee Shang Lun, and Allison Walker. Sokobond. https://www.sokobond.com/, 2014.

9 Tom C. van der Zanden and Hand L. Bodlaender. PSPACE-completeness of Bloxorz and of games with 2-buttons. arXiv:1411.5951, 2014. URL: https://arXiv.org/abs/1411.5951.

10 Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014. Originally appeared at FUN 2012. doi:10.1007/s00224-013-9497-5.

11 Giovanni Viglietta. Lemmings is PSPACE-complete. *Theoretical Computer Science*, 586:120–134, 2015. Originally appeared at FUN 2014. doi:10.1016/j.tcs.2015.01.055.

Taming the Knight's Tour: Minimizing Turns and Crossings

Juan Jose Besa
University of California, Irvine, CA, USA
jbesavi@uci.edu

Timothy Johnson
University of California, Irvine, CA, USA
tujohnso@uci.edu

Nil Mamano
University of California, Irvine, CA, USA
nmamano@uci.edu

Martha C. Osegueda
University of California, Irvine, CA, USA
mosegued@uci.edu

Abstract

We introduce two new metrics of "simplicity" for knight's tours: the number of turns and the number of crossings. We give a novel algorithm that produces tours with $9.5n + O(1)$ turns and $13n + O(1)$ crossings on a $n \times n$ board, and we show lower bounds of $(6 - \varepsilon)n$ and $4n - O(1)$ on the respective problems of minimizing these metrics. Hence, our algorithm achieves approximation ratios of $19/12 + o(1)$ and $13/4 + o(1)$. We generalize our techniques to rectangular boards, high-dimensional boards, symmetric tours, odd boards with a missing corner, and tours for $(1, 4)$-leapers. In doing so, we show that these extensions also admit a constant approximation ratio on the minimum number of turns, and on the number of crossings in most cases.

2012 ACM Subject Classification Human-centered computing → Graph drawings; Theory of computation → Computational geometry; Mathematics of computing → Approximation algorithms

Keywords and phrases Graph Drawing, Chess, Hamiltonian Cycle, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.4

Related Version `https://arxiv.org/abs/1904.02824`

Supplementary Material `https://nmamano.github.io/MinCrossingsKnightsTour/index.html`

Funding The authors were supported by NSF Grant CCF-1616248 and NSF Grant 1815073.

1 Introduction

The game of chess is a fruitful source of mathematical puzzles. The puzzles often blend an appealing aesthetic with interesting and deep combinatorial properties [32]. An old and well-known problem is the knight's tour problem. A *knight's tour* in a generalized $n \times m$ board is a path through all nm cells such that any two consecutive cells are connected by a "knight move" (Fig. 1). For a historic treatment of the problem, see [2]. A knight's tour is *closed* if the last cell in the path is one knight move away from the first one. Otherwise, it is *open*. This paper focuses solely on closed tours, so henceforth we omit the distinction. The knight's tour problem is a special case of the Hamiltonian cycle problem, in which we find a simple cycle that visits all the nodes for a specific class of graphs. These graphs are formed by representing each cell on the board as a node and connecting cells a knight move apart. Existing work focuses on the questions of existence, counting, and construction algorithms.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 4; pp. 4:1–4:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

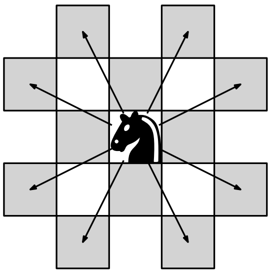


Figure 1 A knight moves two units on one axis and one on the other.

In general, the goal of existing algorithms is to find *any* knight's tour. We propose two new metrics that capture simplicity and structure in a knight's tour. We associate each cell in the board with a point (i, j) in the plane, representing the row and column respectively.

▶ **Definition 1** (Turn). *Given a knight's tour, a* turn *is* a triplet of consecutive cells with non-colinear coordinates.

▶ **Problem 1** (Minimum turn knight's tour). *Finding the knight's tour with the smallest number of turns for a given rectangular $n \times m$ board.*[1]

▶ **Definition 2** (Crossing). *Given a knight's tour, a* crossing *occurs* when the two line segments corresponding to moves in the tour intersect*. E.g. if $\{c_1, c_2\}$ and $\{c_3, c_4\}$ are two distinct pairs of consecutive cells visited along the tour, a crossing happens if the open line segments (c_1, c_2) and (c_3, c_4) intersect.*

▶ **Problem 2** (Minimum crossing knight's tour). *Finding the knight's tour with the smallest number of crossings for a given rectangular $n \times m$ board*[1]

Knight's tours are typically visualized by connecting consecutive cells by a line segment. Turns and crossings make the sequence harder to follow. Minimizing crossings is a central problem in *graph drawing*, the sub-field of graph theory concerned with the intelligible visualization of graphs (e.g., see the survey in [13]). Problem 2 is the natural adaptation for knight's tours. Problem 1 asks for the (self-intersecting) polygon with the smallest number of vertices that represents a valid knight's tour.

1.1 Our contributions

We propose a novel algorithm for finding knight's tours with the following features.

- $9.5n + O(1)$ turns and $13n + O(1)$ crossings on a $n \times n$ board.
- A $19/12 + o(1)$ approximation factoron the minimum number of turns (Problem 1).
- A $13/4 + o(1)$ approximation factor on the minimum number of crossings (Problem 2).
- $O(nm)$ run-time on a $n \times m$ board, i.e., linear to the number of cells, which is optimal.
- The algorithm is fully parallelizable, it can be executed in $O(1)$ time with $O(nm)$ processors in the CREW PRAM model. Since the cell at any given index in the tour sequence (or, conversely, the index of a given cell) can be determined in constant time.

[1] Assuming a knight's tour exists for this board

- It can be generalized to most typical variations of the problem: higher-dimensional cubical boards, rotationally symmetric tours, tours in odd-sized boards that skip a corner cell, and tours for *giraffes*, which move a cell in one dimension and four in another.
- The algorithm can be simulated by hand with ease. This is of particular interest in the context of recreational mathematics and mathematics outreach.

The paper is organized as follows. In Section 1.2, we give an overview of the literature on the knight's tour problem and its variants. We describe the algorithm in Section 2. We prove the approximation ratios in Section 3. We describe how our algorithm extends to related problems in Section 4. We conclude in Section 6.

The tours produced by the algorithm can be generated interactively for different board dimensions at `https://nmamano.github.io/MinCrossingsKnightsTour/index.html`.

1.2 Related Work

Despite being over a thousand years old [32], the knight's tour problem is still an active area of research. We review the key questions considered in the literature.

Existence. In rectangular boards, a tour exists as long as one dimension is even and the board size is large enough; no knight's tour exists for dimensions $1 \times n$, $2 \times n$ or $4 \times n$, for any $n \geq 1$ and, additionally, none exist for dimensions 3×6 or 3×8 [29]. In three dimensions or higher, the situation is similar: a tour exists only if at least one dimension is even and large enough [9, 10, 11]. In the case of open knight's tours, a tour exists in two dimensions if both dimensions are at least 5 [7, 6].

Counting. The number of closed knight's tours in an even-sized $n \times n$ board is at least $\Omega(1.35^{n^2})$ and at most 4^{n^2} [20]. The exact number of knight's tours in the standard 8×8 board is $26, 534, 728, 821, 064$ [23]. Furthermore algorithms for enumerating multiple [30] and enumerating all [1] knight's tours have also been studied.

Algorithms. Greedy algorithms have been popular in tour construction. Usually doing so by selecting one step at a time according to a heuristic rule. Historically, greedy algorithms have been popular. The idea is to construct the tour in order, one step at a time, according to some heuristic. Warnsdorff's rule and its refinements [27, 1, 31] work well in practice for small boards, but do not scale to larger boards [25]. The basic idea is to choose the next node with fewest continuations, which is also useful for the Hamiltonian cycle problem [27].

To our knowledge, all efficient algorithms for arbitrary board sizes before this paper are based on a divide-and-conquer approach. The tour is solved for a finite set of small, constant-size boards. Then, the board is covered by these smaller tours like a mosaic. The small tours are connected into a single one by swapping a few knight moves. This can be done in a bottom-up [29, 6, 9, 10, 16] or a top-down recursive [26, 21] fashion. This process is simple and can be done in time linear on the number of cells and, like ours, are also highly parallelizable [6, 26] since they are made of repeating patterns.

Divide-and-conquer is not suitable for finding tours with a small number of turns or crossings. Since each base solution has constant size, a $n \times n$ board is covered by $\Theta(n^2)$ of them, and each one contains turns and crossings. Thus, the divide-and-conquer approach necessarily results in $\Theta(n^2)$ turns and crossings. In contrast, our algorithm has $O(n)$.

Extensions. The above questions have been considered in related settings. Extensions can be classified into three categories, which may overlap:

- **Tours with special properties.** Our work can be seen as searching for tours with special properties. *Magic knight's tours* are also in this category: tours such that the indices of each cell in the tour form a magic square (see [3] for a survey).
  The study of symmetry in knight's tours dates back at least to 1917 [4]. Symmetric tours under 90 degree rotations exist in $n \times n$ tours where $n \geq 6$ and n is of the form $4k+2$ for some k [8]. Parberry extended the divide-and-conquer approach to produce tours symmetric under 90 degree rotations [26]. Jelliss provided results on which rectangular board sizes can have which kinds of symmetry [15].
  Both of our proposed problems are new, but minimizing crossings is related to the *uncrossed knight's tour problem*, which asks to find the longest sequence of knight moves without *any* crossings [34]. This strict constraint results in incomplete tours. This problem has been further studied in two [14, 12] and three [19] dimensions.
- **Board variations.** Besides higher dimensions, knight's tours have been considered in other boards, such as torus boards, where the top and bottom edges are connected, and the left and right edges are also connected. Any rectangular torus board has a closed tour [33]. Another option is to consider boards with odd width and height. Since boards with an odd number of cells do not have tours, it is common to search for tours that skip a specific cell, such as a corner cell [26].
- **Move variations.** An (i, j)-*leaper* is a generalized knight that moves i cells in one dimension and j in the other (the knight is a $(1, 2)$-leaper) [24]. Knuth studied the existence of tours for general (i, j)-leapers in rectangular boards [18]. Tours for *giraffes* ($(4, 1)$-leapers) were provided in [8] using a divide-and-conquer approach. Chia and Ong [5] study which board sizes admit generalized (a, b)-leaper tours. Kamčev [16] showed that any board with sufficiently large and even size admits a $(2, 3)$-, $(2, 5)$-, and a $(a, 1)$-leaper tour for any even a, and generalized this to any higher dimensions. Note that a and b are required to be coprime and not both odd, or no tour can exist [16].

2 The Algorithm

Given that one of the dimensions must be even for a tour to exist, we assume, without loss of generality, that the width w of the board is even, while the height h can be odd. We also assume that $w \geq 16$ and $h \geq 12$. The construction still works for some smaller sizes, but may require tweaks to its most general form described here.

Quartet moves. What makes the knight's tour problem challenging is that knight jumps leave "gaps". Our first crucial observation is that a quartet of four knights arranged in a square 2×2 formation can move "like a king": they can move horizontally, vertically, or diagonally without leaving any gaps (Figure 2).

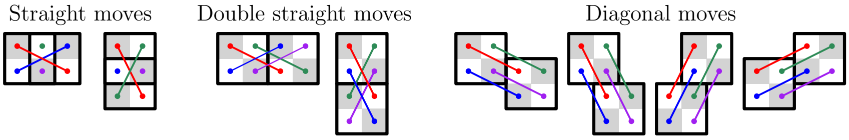


Figure 2 Quartet of knights moving in unison leaving no unvisited squares. In a straight move, the starting and ending position of the quartet overlap because two knights remain in place.

Algorithm 1 Knight's tour algorithm for even width $w \geq 16$ and height $h \geq 12$.

```
1. Fill the corners of the board as follows:
Bottom-left: first junction in Figure 4.
Top-right: junction of height $5 + ((w/2 + h - 1) \bmod 4)$ in Figure 4 except the first one.
Bottom-right: Sequence $(w/2 + 2) \bmod 4$ in Figure 5.
Top-left: Sequence $(3 - h) \bmod 4$ in Figure 5 rotated 180 degrees.
2. Connect the four corners using formation moves, by moving along diagonals from the
bottom-left corner to the top-right corner as in Figure 3. To transition between diagonals:
Vertical edges: use a double straight up move (Figure 2).
Horizontal edges: use Sequence 1 in Figure 5.
```

By using the "formation moves" depicted in Figure 2, four knights can easily cover the board moving vertically and horizontally while remaining in formation. Of course, the goal is to traverse the entire board in a single cycle, not four paths. We address this issue with special structures placed in the bottom-left and top-right corners of the board, which we call *junctions*, and which tie the paths together to create a single cycle. Note that using only straight formation moves leads to tours with a large number of turns and crossings. Fortunately, two consecutive diagonal moves in the same direction introduces no turns or crossings, so our main idea is to use as many diagonal moves as possible. This led us to the general pattern shown in Figure 3.

The full algorithm is given in Algorithm 1. The formation starts at a junction at the bottom-left corner and ends at a junction at the top-right corner. To get from one to the other, it zigzags along an odd number of parallel diagonals, alternating between downward-right and upward-left directions. The junctions in Figure 4 have a *height*, which influences the number of diagonals traversed by the formation.

At the bottom-left corner, we use a junction with height 5. At the top-right corner, we use a junction with height between 5 and 8. Choosing the height as in Algorithm 1 guarantees that, for any board dimensions, an odd number of diagonals fit between the two junctions. Sequence 1 in Figure 5, which we call the *heel*, is used to transition between diagonals along the horizontal edges of the board. The two non-junction corners require special sequences of quartet moves, as depicted in Figure 5. Sequences $1, 2, 3$, and 0 are used when the last heel ends $0, 2, 4$, and 6 columns away from the vertical edge, respectively. These variations and the top-right junction are *predictable* because they cycle as the board dimensions grow, so in Algorithm 1 we give expressions for them in terms of w and h.

2.1 Correctness

It is clear that the construction visits every cell, and that every node in the underlying graph of knight moves has degree two. However, it remains to be argued that the construction is actually a single closed cycle. For this, we need to consider the choice of junctions.

A junction is a pair of disjoint knight paths whose four endpoints are adjacent as in the quartet formation. Thus, the bottom-left junction connects the knights into two pairs. Denote the four knight positions in the formation by tl, tr, bl, br, where the first letter indicates top/bottom and the second left/right. We consider the three possible *positional matchings* with respect to these positions: horizontal matching $H = (tl, tr), (bl, br)$, vertical matching $V = (tl, bl), (tr, br)$, and cross matching $X = (tl, br), (tr, bl)$. Let $\mathcal{M} = \{H, V, X\}$ denote the set of positional matchings. We are interested in the effect of formation moves on the positional matching. A formation move does not change which knights are matched with which, but a non-diagonal move changes their positions, and thus their matchings. For instance, a horizontal matching becomes a cross matching after a straight move to the right.

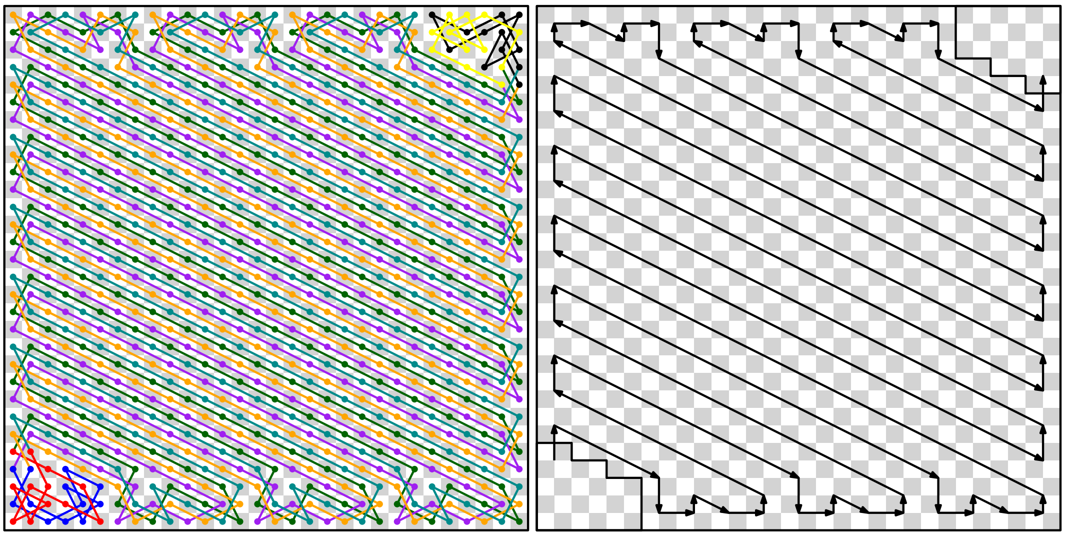


Figure 3 Side by side comparison between the knight's tour **(left)** and the underlying quartet moves **(right)** in a 30 × 30 board. Arrows illustrate sequences of consecutive formation moves. Starting from the bottom-left square of the board, the single knight's tour follows the colored sections in the order: red, green, yellow, purple, blue, orange, black, cyan, and back to red.

It is easy to see that a straight move upwards or downwards has the same effect on the positional matching. Similarly for left and right straight moves. Thus, we classify the formation moves in Figure 2 (excluding double straight moves, which are a composition of two straight moves) into vertical straight moves ↕, horizontal straight moves ↔, and diagonal moves ⤢. Let $\mathcal{S} = \{$↕, ↔, ⤢$\}$ denote the three types of quartet moves. We see each move type $s \in \mathcal{S}$ as a function $s : \mathcal{M} \to \mathcal{M}$ (see Table 1). Note that the diagonal move ⤢ is just the identity. Given a sequence of moves $S = (s_1, \dots, s_k)$, where each $s_i \in \mathcal{S}$, let $S(M) = s_1 \circ \cdots \circ s_k(M)$. The move types ↕, ↔, ⤢ seen as functions are, in fact, permutations (Table 1). It follows that *any* sequence of formation moves permutes the positional matchings, according to the composed permutation of each move in the sequence. There are six possible permutations of the three positional matchings, three of which correspond to the "atomic" formation moves ⤢, ↕, and ↔. The other three permutations can be obtained by composing atomic moves, for instance, with the compositions ↕↔, ↔↕, and ↕↔↕ (Table 1). Thus, any

Table 1 Result of applying each type of formation move, as well as three compositions of sequences of moves, to each formation matching.

	⤢	↕	↔	↕↔	↔↕	↕↔↕
V	V	X	V	H	X	H
H	H	H	X	X	V	V
X	X	V	H	V	H	X

Table 2 Cayley table for the group of positional matching permutations.

	⤢	↕	↔	↕↔	↔↕	↕↔↕
⤢	⤢	↕	↔	↕↔	↔↕	↕↔↕
↕	↕	⤢	↕↔	↔	↕↔↕	↔↕
↔	↔	↔↕	⤢	↕↔↕	↕	↕↔
↕↔	↕↔	↕↔↕	↕	↔↕	⤢	↔
↔↕	↔↕	↔	↕↔↕	⤢	↕↔	↕
↕↔↕	↕↔↕	↕↔	↔↕	↕	↔	⤢

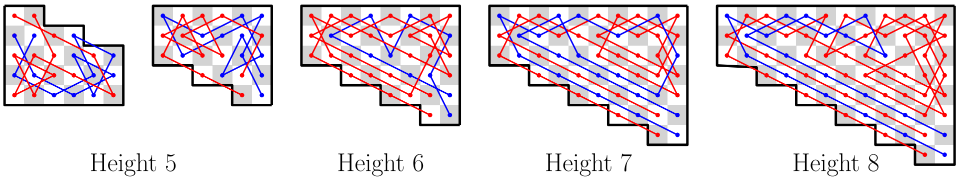

Height 5 Height 6 Height 7 Height 8

Figure 4 Junctions used in our construction.

sequence of moves permutes the positional matchings in the same way as one of the sequences in the set {⤢, ↕, ↔, ↕↔, ↔↕, ↕↔↕}. This is equivalent to saying that this set, under the composition operation, is isomorphic to the symmetric group of degree three. Table 2 shows the Cayley table of this group.

Let $T_{w,h}$ be the sequence of formation moves that goes from the bottom-left junction to the top-right one in Algorithm 1 in a $w \times h$ board.

▶ **Lemma 3.** *For any even $w \geq 16$ and any $h \geq 12$, $T_{w,h}(H) = H$.*

Proof. We show that the entire sequence of moves $T_{w,h}$ is either neutral or equivalent to single vertical move, depending on the board dimensions. We refer to a sequence of moves as *neutral* when the knights have the same formation at the beginning and end. sAccording to Table 1, this suffices to prove the lemma.

The sequence $T_{w,h}$ consists mostly of diagonal moves, which are neutral. The transition between diagonals along the vertical edges consist of two vertical moves, which are also neutral (↕↕= ⤢). The heel is also neutral, as it consists of the sequence ↕↕↔↔↕↔↔↕ (omitting diagonal moves) which is again equivalent to ⤢. This is easy to see by noting that any two consecutive vertical or horizontal moves cancel out. It is depicted in detail in Figure 6. Thus, $T_{w,h}$ reduces to composing the sequences in the bottom-right and top-left corners. As mentioned, Sequence 1 (the heel) is neutral. It is easy to see that the other sequences (counting each part of Sequence 2 separately) is equivalent to ↕. Thus, we get that $T_{w,h}$ is simply the composition of zero to four vertical moves, depending on the width and height of the board. This further simplifies to zero or one vertical moves. ◀

From these it is easy to see that the algorithm produces a correct tour, however we proceed with a more detail proof.

▶ **Theorem 4** (Correctness). *Algorithm 1 outputs a valid knight's tour in any board with even width $w \geq 16$ and with height $h \geq 12$.*

Proof. Clearly, the formation moves in our construction yield a set of disjoint cycles in the underlying knight-move graph. We prove that after attaching them to the junctions they actually form one cycle. Given a set of disjoint cycles in a graph, *contracting* a node in one of the cycles is the process of removing it and connecting its two neighbors in the cycle. Contracting a node in a cycle of length ≥ 3 does not change the number of cycles. Thus, consider the remaining graph if we contract all the nodes except the four endpoints of the top-right junction.

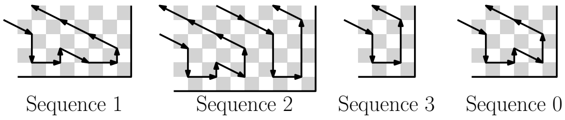

Sequence 1 Sequence 2 Sequence 3 Sequence 0

Figure 5 The four possible cases for the bottom-right corner.

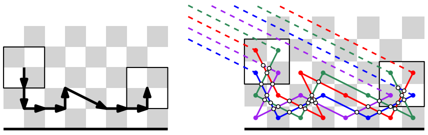


Figure 6 Visualization of the heel: permutation of knights' positions (in color) and its 32 crossings (white disks).

Note that we use a horizontal matching in the bottom-left junction and a vertical matching in the top-right junction. Contracting the non-endpoint nodes inside the top-right junction leaves the two edges corresponding to the vertical matching. By Lemma 3, contracting the nodes outside the top-right junction leaves the edges corresponding to a horizontal matching. Thus, the resulting graph is a single cycle of four nodes. ◀

3 Lower Bounds and Approximation Ratios

In this section, we analyze the approximation ratio that our algorithm achieves for Problem 1 and Problem 2. For simplicity, we restrict the analysis to square boards. Furthermore, it is worth noting that the way in which the input is encoded affects the complexity analysis of the problem, we have included a thorough discussion of these in Appendix A

3.1 Number of Turns

All the turns in our construction happen near the edges. The four corners account for a constant number of turns. The left and right edges have eight turns for each four rows. As it can be seen in Figure 6, the heel has 22 turns, so the top and bottom edges have 22 turns each for each eight columns. Therefore, the number of turns in our construction is bounded by $2\frac{8}{4}n + 2\frac{22}{8}n + O(1) = 9.5n + O(1)$.

Lower bound. First, note that every cell next to an edge *must* contain a turn. This accounts for $4n - 4$ turns. A simple argument, sketched in Appendix B, improves this to a $4.25n - O(1)$ lower bound. Here we focus on the main result, a lower bound of $(6 - \varepsilon)n$ for any $\varepsilon > 0$. We start with some intermediate results.

We associate each cell in the board with a point (i, j) in the plane, where i is the row of the cell and j is the column. An edge cell only has four moves available. We call the directions of these moves D_1, D_2, D_3, and D_4, in clockwise order. For an edge cell c, let $r_i(c)$, with $1 \leq i \leq 4$, denote the ray starting at c and in direction D_i. That is, the ray that passes through the cells reachable from c by moving along D_i.

Let a and b be two cells along the left edge of the board, with a above b. The discussion is symmetric for the other three edges. Given two intersecting rays r and r', one starting from a and one from b, let $S(r, r')$ denote the set of cells in the region of the board *bounded* by r and r': the set of cells below or on r and above or on r'. We define the *crown* of a and b as the following set of cells (see Figure 7):

$$\text{crown}(a, b) = S(r_2(a), r_1(b)) \cup S(r_3(a), r_2(b)) \cup S(r_4(a), r_3(b)).$$

We can associate, with each edge cell c, the two maximal sequences of moves without turns in the tour that have c as an endpoint. We call them the *legs* of c. We say that legs begin at c and end at their other endpoint. We say two legs of different cells *collide* if they end at the same cell. Let $C_{a,b}$ denote the set of edge cells along the right edge between a and b (a and b included). The following is easy to see.

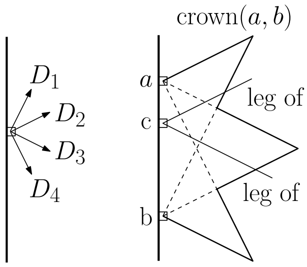


Figure 7 Terminology for the lower bound. Note that c is a clean cell (with respect to the crown of a and b) because both of its legs escape it.

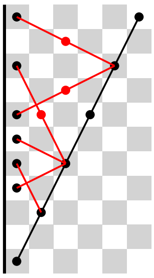


Figure 8 The black leg collides would collide with all the red legs.

▶ Remark 5. Any collision between the legs of edge cells in $C_{a,b}$ happens inside crown(a, b).

We say that a leg of a cell in $C_{a,b}$ *escapes* the crown of a and b if it ends outside the crown. We say an edge cell in $C_{a,b}$ is *clean*, with respect to $C_{a,b}$, if both of its legs escape. We use the following observation, to later obtain the lower bound and show that there is only a constant number of clean cells inside a crown.

▶ Remark 6. Let $m = |C_{a,b}|$ and k be the number of clean cells in $C_{a,b}$. The number of turns inside crown(a, b) is *at least* $m + (m - k)/2$.

Proof. Each edge cell is one turn. Further, each of the $m - k$ non-clean cells have a leg that ends in a turn inside the crown. This turn may be because it collided with the leg of another edge cell in the crown. Thus, there is at least one turn for each two non-clean edge cells. ◀

▶ **Lemma 7.** *Let a, b be two cells along the left edge of the board, with a above b. There are at most 122 clean cells inside crown(a, b).*

Proof. First we show that there are at most 60 clean cells such that one of their legs goes in direction D_1. For the sake of contradiction, assume that there are at least 61. Then, there are two, c and d, such that c is $60r$ rows above d, for some $r \in \mathbb{N}, r \geq 1$. The contradiction follows from the fact that the other leg of c, which goes along D_2, D_3, or D_4, would collide with the leg of b along D_1. This is because, for any $l \geq 1$, the leg of b along D_1 collides with (see Figure 8):

- any leg along D_2 starting from a cell $3l$ rows above b,
- any leg along D_3 starting from a cell $5l$ rows above b, and
- any leg along D_4 starting from a cell $4l$ rows above b.

Since $60r$ is a multiple of $3, 4$, and 5, no matter what direction the other leg of c goes, it collides with the leg of d. As observed, this collision happens inside the crown. Thus, c and d are not clean. By a symmetric argument, there are at most 60 clean cells such that one of their legs goes in direction D_4.

Finally, note that there can only be two clean cells with legs in D_2 *and* D_3. This is because, by a similar argument, there cannot be two such cells an even distance of each other; the leg along D_3 of the top one would collide against the leg along D_2 of the bottom one. ◀

▶ **Corollary 8.** *Suppose that the crown of a and b has $m \geq 122$ edge cells. Then, there are at least $(m - 122)/2$ turns inside the crown at non-edge cells.*

Now, consider the iterative process depicted in Figure 9, defined over the unit square. The square is divided in four sectors along its main diagonals. Whereas earlier we used the term "crown" to denote a set of cells, here we use it to denote the polygon with the *shape* of

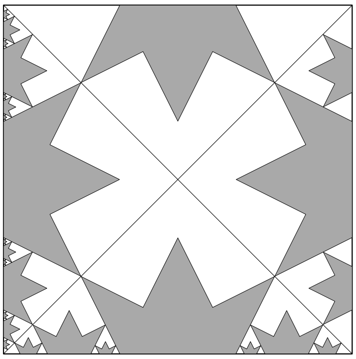


Figure 9 Each sector of the square shows the process after a different number of iterations: $1, 2, 3$, and 4 iterations on the top, right, bottom, and left sectors, respectively.

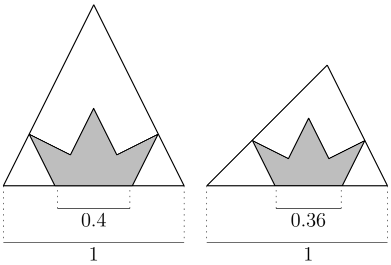


Figure 10 Lower bounds on two ratios.
Left: the ratio between the gap between consecutive crowns and the base of the maximum-size crown that fits in the gap is > 0.4.
Right: the ratio between the gap between a crown and a main diagonal and the base of the maximum-size crown that fits in the gap is > 0.36.

a crown. On the first step, a maximum-size crown is placed on each sector. At step $i > 1$, we place 2^{i-1} more crowns in each sector. They are maximum-size crowns, subject to being disjoint from previous crowns, in each gap between previous crowns and between the crowns closest to the corners and the main diagonals.

▶ **Lemma 9.** *For any $1 > \varepsilon > 0$, there exists an $i \in \mathbb{N}$ such that at least $(1 - \varepsilon)$ of the boundary of the unit square is inside a crown after i iterations of the process.*

Proof. At each iteration, a constant fraction larger than 0.36 of the length on each side that is not in a crown is added to a new crown (Figure 10). This gives rise to a series A_i for the fraction of the side inside crowns after i iterations: $A_1 = 1/3$, $A_{i+1} > A_i + 0.36(1 - A_i)$ for $i > 1$; this series converges to 1. ◀

▶ **Theorem 10** (Lower bound). *For any constant $\varepsilon > 0$, there is a sufficiently large n such that any knight's tour on a $n \times n$ board requires $(6 - \varepsilon)n$ turns.*

Proof. We show a seemingly weaker form of the claim: that there is a sufficiently large n such that any knight's tour on a $n \times n$ board requires $(6 - 2\varepsilon)n - C_\varepsilon$ turns, where C_ε is a constant that depends on ε but not on n. This weaker form is in fact equivalent because, for sufficiently large n, $C_\varepsilon < \varepsilon n$, and hence $(6 - 2\varepsilon)n - C_\varepsilon > (6 - 3\varepsilon)n$. Thus, the claim is equivalent up to a multiplicative factor in ε, but note that it is a claim about arbitrarily small ε, so it is not affected by a multiplicative factor loss.

Let i be the smallest number of iterations of the iterative process in Figure 9 such that at least $(1 - \varepsilon)$ of the boundary of the unit square is inside crown shapes. The number i exists by Lemma 9. Fix S to be the corresponding set of crown shapes, and $r = |S|$. Note that $r = 4(2^i - 1)$ is a constant that depends only on ε. Now, consider a square $n \times n$ board with the crown shapes in S overlaid in top of them. Let the board size n be such that the smallest crown in S contains more than 122 edge cells. Then, by Corollary 8, adding up the turns at non-edge cells over all the crowns in S, we get at least $4n(1 - \varepsilon)/2 - 61r$ turns. Adding the $4n - 4$ turns at edge cells, we get that the total number of turns is at least $(6 - 2\varepsilon)n - 61r - 4$. To complete the proof, consider $C_\varepsilon = 61r + 4$. ◀

▶ **Corollary 11.** *Algorithm 1 achieves a $19/12 + o(1)$ approximation on the minimum number of turns.*

Proof. In a $n \times n$ board, let $ALG(n)$ denote the number of turns in the tour produced by Algorithm 1 and $OPT(n)$ denote the minimum number of turns. Let $\varepsilon > 0$ be an arbitrarily small constant. We show an n_0 exists such that $\forall n \geq n_0$, $ALG(n)/OPT(n) < 19/12 + \varepsilon$. As mentioned, for any even $n \geq 16$, $ALG(n) < 9.5n + c$ for some small constant c. In addition, by Theorem 10, for sufficiently large n, $OPT(n) > (6 - \varepsilon)n$. Thus, $ALG(n)/OPT(n) < (9.5n + c)/(n(6 - \varepsilon))$ Furthermore, for large enough n, $c/((6 - \varepsilon)n) < \varepsilon/2$, so

$$\frac{ALG(n)}{OPT(n)} < \frac{9.5}{6-\varepsilon} + \frac{\varepsilon}{2} = \frac{19}{12-2\varepsilon} + \frac{\varepsilon}{2} < \frac{19+6\varepsilon}{12} + \frac{\varepsilon}{2} = \frac{19}{12} + \varepsilon. \qquad ◀$$

3.2 Number of Crossings

Similarly to the case of turns, all the crossings in our construction happen near the edges. The four corners account for a constant number of crossings. The left and right edges have 10 crossings for each four rows. The top and bottom edges have 32 crossings for each eight columns (Figure 6). Therefore, the number of turns in our construction is bounded by $2\frac{10}{4}n + 2\frac{32}{8}n + O(1) = 13n + O(1)$.

▶ **Lemma 12.** *Any knight's tour on an $n \times n$ board has at least $4n - O(1)$ crossings.*

Proof. Let T be an arbitrary knight's tour on an $n \times n$ board. We show that T has $n - O(1)$ crossings involving knight moves incident to the cells along the left edge of the board. An analogous argument holds for the three other edges of the board, which combined yield the desired bound.

We partition the edge cells along the left-most column into sets of three consecutive cells, which we call *triplets* (if n is not multiple of three, we ignore any remaining cells, as they only account for a constant number of crossings). Two triplets are *adjacent* if they contain adjacent cells. Each triplet has six associated knight moves in the tour T, two for each of its cells. We call the choice of moves the *configuration* of the triplet. Since there are $\binom{4}{2} = 6$ choices of moves for each cell, there are $6^3 = 216$ possible configurations of each triplet.

Consider a weighted directed graph G with a node for each of the 216 possible triplet configuration and an edge from every node to every node, including a loop from each node to itself. The graph has weights on both vertices and edges. Given a node v, let $C(v)$ denote its associated configuration. The weight of v is the number of crossings between moves in $C(v)$. The weight of each edge $v \to u$ is the number of crossings between moves in $C(v)$ and moves in $C(u)$ when $C(v)$ is adjacent and above $C(u)$.

Each path in G represents a choice of move configurations for a sequence of consecutive triplets. Note that if two knight moves in T with endpoints in edge cells cross, the edges cells containing the endpoints are either in the same triplet or in adjacent triplets. Thus, the sum of the weights of the vertices and edges in the path equals the total number of crossings

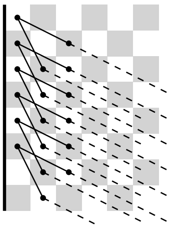


Figure 11 The configuration pattern along the board's edge with the minimum number of crossings. Triplet configurations (solid) and their continuation (dashed) without extra crossings.

among all of these moves. Since G is finite, any sufficiently long path eventually repeats a vertex. Given a cycle c, let $w(c)$ be the sum of the weights of nodes and edges in c, divided by the length of c. Let c^* be the cycle in G minimizing w. Then, $w(c^*)$ is a lower bound on the number of crossings per triplet along to edge.

By examining G, we can see that $w(c^*) = 3$. Figure 11 shows an optimum cycle, which in fact uses only one triplet configuration. The cycle minimizing w can be found using Karp's algorithm for finding the minimum mean weight cycle in directed graphs [17], which runs in $O(|V| \cdot |E|)$ time in a graph with vertex set V and edge set E. However, this requires modifying the graph G, as Karp's algorithm is not suitable for graphs that also have node weights. We transform G into a directed graph G' with weights on only the edges and which preserves the optimal solution, as follows. We double each node v in G into two nodes v_{in}, v_{out} in G'. We also add an edge $v_{in} \to v_{out}$ in G' with weight equal to the weight of v in G. For each edge $v \to u$ in G, we add an edge $v_{out} \to u_{in}$ in G'. ◀

Since we only counted crossings between moves incident to the first column, a question arises of whether the lower bound can be improved by considering configurations spanning more columns (e.g., the two or three leftmost columns). The answer is negative for any constant number of columns. Figure 11 shows that the edges can be extended to paths that cover any fixed number of rows away from the edge without increasing the number of crossings.

▶ **Corollary 13.** *Algorithm 1 achieves a $13/4 + o(1)$ approximation on the minimum number of crossings.*

4 Extensions

The idea of using formation moves to cover the board and junctions to close the tour is quite robust to variations of the problem. We show how this can be done in some of the most popular generalizations of the problem.

A variant that we do not consider is torus boards (where opposite edges are connected). The problem of finding tours with a small number of turns seems easier on a torus board, because one is not forced to make a turn when reaching an edge. Nonetheless, in a square $n \times n$ torus board, $\Omega(n)$ turns are still required, because making n consecutive moves in the same direction brings the knight back to the starting position, so at least one turn is required for each n visited cells. The tours for torus boards in [33] match this lower bound up to constant factors, at least for some board dimensions (see the last section in [33]). Crossings are not straightforward to define in torus boards.

4.1 High-dimensional boards

We extend our technique to three and higher dimensions. In d dimensions, a knight moves by 1 and 2 cells along any two dimensions, and leaves the remaining coordinates unchanged. A typical technique to extend a knight's tour algorithm to three dimensions is the "layer-by-layer" approach [32, 9, 10]: a 2D tour is reused on each level of the 3D board, adding the minimal required modifications to connect them into a single tour. We also follow this approach. (Watkins and Yulan [28] consider a generalizations of knight moves where the third coordinate also has a positive offset, but this is not as common.) For illustration purposes, we start with the 3D case, and later extend it to the general case. We require one dimenson to be even and ≥ 16 and another dimension to be ≥ 12, which we assume w.l.o.g. to be the first two. The rest can be any size. Note that at least one dimension must be even, or no tour exists [11].

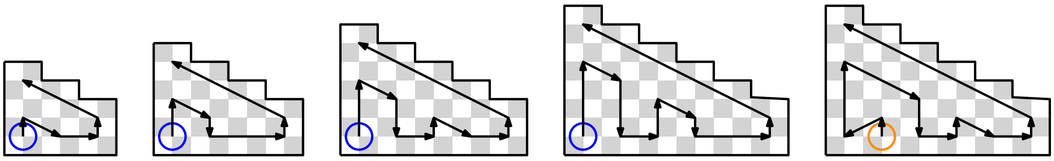


Figure 12 Corners where the knights stay in formation and end at specific positions.

The construction works as follows. The 2D construction is reused at each level. However, there are only two actual junctions, one on the first layer, of height 5, and one on the last layer, which may have any of the four heights in Figure 4. Every other junction is replaced by a sequence of formation moves. At every layer except the last, the formation ends adjacent to the corner using one of the sequences of moves in Figure 12 (note that we show sequences for 4 different heights, thus guaranteeing that one shape fits for any board dimensions). The layers are connected with a formation move one layer up and two cells to the side, as in Figure 14. At every layer except the first, the formation starts with the rightmost sequence of moves in Figure 12. A full example is illustrated in Figure 13.

Note that, since the sequence of moves between junctions is more involved than in two dimensions, Lemma 3 may not hold. There is, however, an easy fix: if the entire sequence is not a single cycle, replace the first junction with one that has a vertical matching (second junction in Figure 4, rotate 180 degrees). This then makes a cycle.

If the number of dimensions is higher than three, simply observe that the same move used between levels can also be used to jump to the next dimension; instead of changing by 1 the third coordinate, change the fourth. After the first such move, the formation will be at the "top" of the second 3D board, which can be traversed downwards. This can be repeated any number of times, and generalizes to any number of dimensions.

Note that in a n^d board, $\Omega(n^{d-1})$ turns are needed, because there are n^d cells and a turn must be made after at most $n/2$ moves. Note that our construction has $O(n^{d-1})$ turns, as it consists of n^{d-2} iterations of the 2D tour. Thus, it achieves a constant approximation ratio on the minimum number of turns. We do not know of any lower bound on the number of crossings in higher dimensions.

4.2 Odd boards

We show how to construct a tour for a 2D board with odd dimensions which visits every cell except a corner cell. This is used in the next section to construct a tour that is symmetric under 90° rotations.

Let the board dimensions be $w \times h$, where $w > 16$ and $h > 12$ are both odd. First, we use Algorithm 1 to construct a $(w-1) \times h$ tour which is missing the leftmost column. Then, we extend our tour to cover this column, except the bottom cell, with the variations of our construction depicted in 15. In particular, for the top-left corner, recall that we use sequence $(3-h) \bmod 4$ in Figure 5. Here, the height h is odd, so we only need adaptations for Sequences 2 and 0.

4.3 90 Degree Symmetry

In this section, we show how to construct a symmetric tour under 90 degree rotations. We say a tour is symmetric under a given geometric operation if the tour looks the same when the operation is applied to the board.

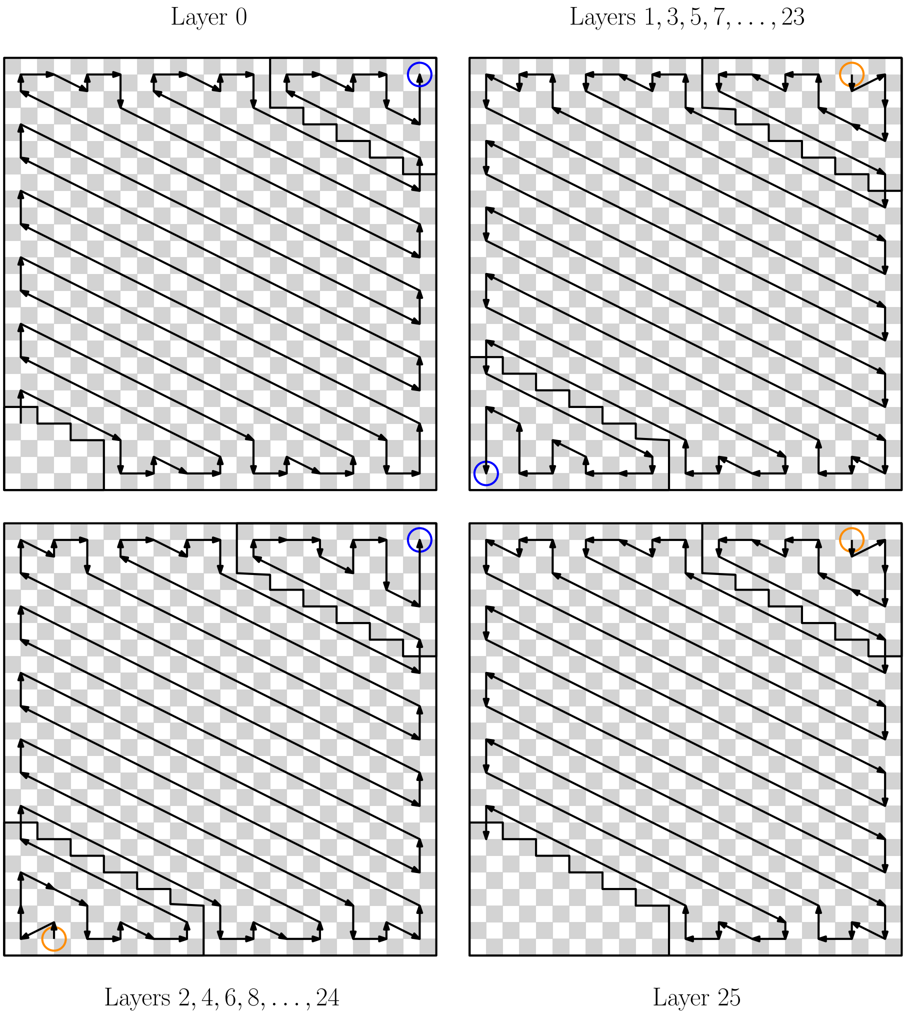

Layer 0
Layers $1, 3, 5, 7, \ldots, 23$
Layers $2, 4, 6, 8, \ldots, 24$
Layer 25

Figure 13 Quartet moves for a $3D$ tour in a $26 \times 26 \times 26$ board. The quartet can move from the blue circle at each layer to the orange circle in the next layer by a quartet move.

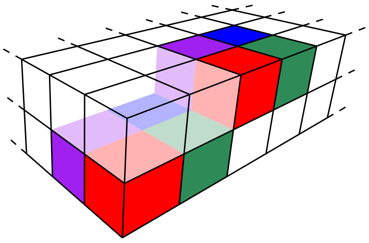


Figure 14 Formation move across layers. Each color shows the starting and ending position of one of the knights.

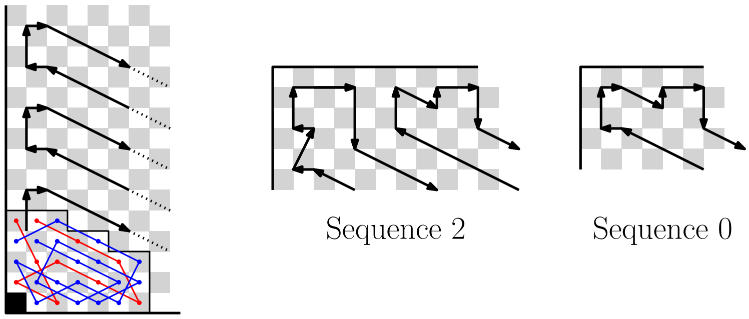


Figure 15 Adaptations required to add a row to the left of the normal construction, with a missing cell in the junction.

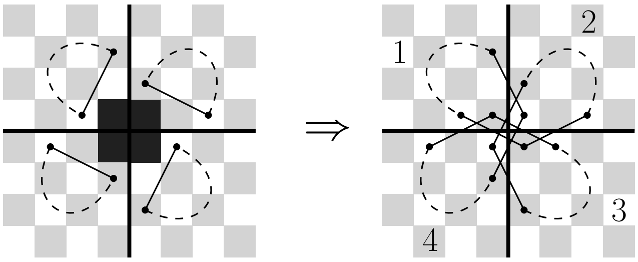


Figure 16 This transformation appears in [26]. **Left:** four tours missing a corner square and containing a certain edge. The dashed lines represent the rest of the tour in each quadrant, which cover every square except the dark square. **Right:** single tour that is symmetric under 90° rotations. The numbers on the right side indicate the order in which each part of the tour is visited, showing that the tour is indeed a single cycle.

As a side note, our construction is already nearly symmetric under 180° rotations. For board dimensions such as 30×30 where opposite corners have the same shape, the only asymmetry is in the internal wiring of the junctions. However, the construction cannot easily be made fully symmetric. It follows from the argument in the proof of Lemma 3 that if the two non-junction corners are equal, the entire sequence of formation moves from one junction to the other is neutral. Thus, using the same junction in both corners, as required to have symmetry, would result in two disjoint cycles.

Symmetric tours under 90° rotations exist only for square boards where the size $n = 4k+2$ is a multiple of two but not a multiple of four [8]. In [26], Parberry gives a construction for knight's tours missing a corner cell and then shows how to combine four such tours into a single tour symmetric under 90° rotations. We follow the same approach to obtain a symmetric tour with a number of turns and crossings linear on n, and thus constant approximation ratios.

In our construction from Section 4.2, cell $(0,0)$ is missing, and edge $e = \{(0,1),(2,0)\}$ is present. This suffices to construct a symmetric tour. Divide the $2n \times 2n$ board into four equal quadrants, each of which is now a square board with odd dimensions. Use the construction for odd bords to fill each quadrant, rotated so that the missing cell is in the center. Finally, connect all four tours as in Figure 16.

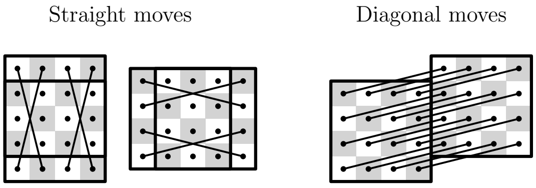


Figure 17 Formation of 16 giraffes moving together without leaving any unvisited squares.

5 Giraffe's tour

A giraffe is a leaper characterized by the move $(1, 4)$ instead of $(1, 2)$. Giraffe's tours are known to exist on square boards of even size $n \geq 104$ [16] and on square boards of size $2n$ when n is odd and bigger than 4 [8]. Our result extends this to some rectangular sizes.

We adapt our techniques for finding giraffe's tours with $O(w + h)$ turns and crossings, where w and h are the width and height of the board. We use a formation of 4×4 giraffes. Figure 17 shows the formation moves, Figure 18 shows the analogous of a heel to be used to transition between diagonals, and Figure 19 shows the two junctions. Figure 20 shows how these elements are combined to cover the board.

We restrict our construction to rectangular boards where $w = 32k + 20$, for some $k \geq 1$, and $h = 8l+14$, for some $l \geq 1$ (extending the results to more boards would require additional heel variations).

We start at the bottom-left junction as in the knight's tour. We transition between diagonals along the bottom edge with a giraffe heel, and along the top edge with a flipped giraffe heel. We transition between diagonals along the left and right edges with four consecutive upward moves. The junction has width 20 and each heel has width 32, so there are k heels along the bottom and top edges. The junction has height 11 and the tip of the heel has height 3, so there are l sequences of four upward moves along each side (see Figure 20).

It is easy to see that the construction visits every cell. As in the case of knight's tours, for the result to be a valid giraffe's tour it should be a cycle instead of a set of disjoint cycles. Note that the matchings in the two junctions form a cycle. Thus, if the formation reaches the top-right junction in the same matching as they left the bottom-left junction, the entire construction is a single cycle (by an argument analogous to Theorem 4).

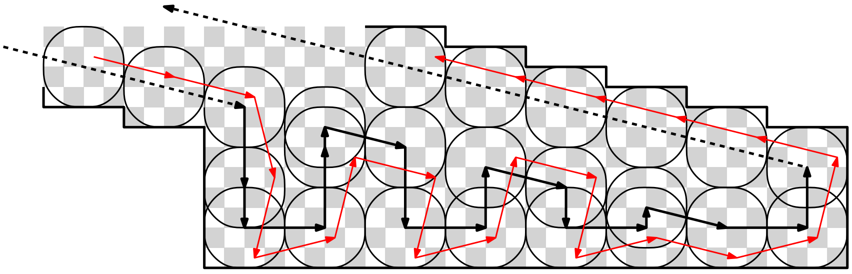


Figure 18 A giraffe heel. The formation moves are shown with black arrows (grouping up to four sequential straight moves together) The intermediate positions of the formation are marked by rounded squares, showing that every cell is covered. Note that the tip of the heel fits tightly under the next heel. The red line shows the path of one specific giraffe.

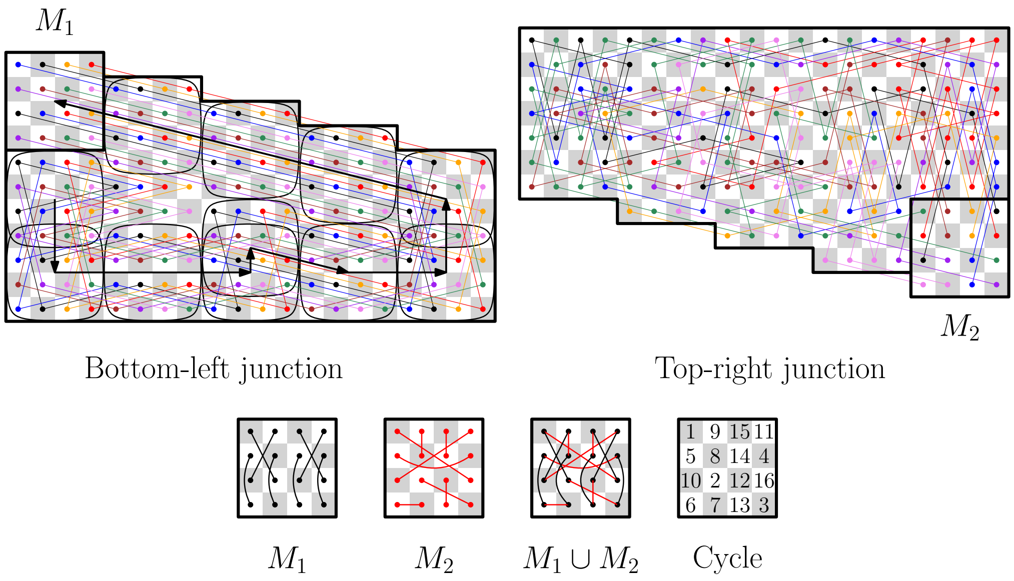


Figure 19 Two giraffe junctions, their corresponding matchings, and the union of their matchings. The bottom-left junction consists mostly of formation moves, whereas the top-right one was computed via brute-force search. The cycle through the edges of the union is shown with the index of each node.

Let H, F, and U denote the sequences of formation moves in the heel, in the flipped heel, and the sequence of four upward moves, respectively. Let $T_{w,h}$ denote the entire sequence of moves from one junction to the other, where $w = 32k + 20$, for some $k \geq 1$, and $h = 8l + 14$, for some $l \geq 1$. Note that $T_{w,h}$ is a concatenation, in some order, of H k times, F k times, and U $2l$ times (we can safely ignore diagonal moves, which do not change the coordinates of the giraffes within the formation). Let M be the matching of the bottom-left junction. We want to argue that, after all the moves in $T_{w,h}$, the giraffes are still in matching M, that is, $T_{w,h}(M) = M$ using the notation from Section 2.

We show that not only the giraffes arrive to the other junction in the same matching but, in fact, they arrive in the same coordinates in the formation as they started. First, note that U has the effect of flipping column 1 with 2 and column 3 with 4 in the formation. Perhaps surprisingly, H and F have the same effect. This is tedious but can be checked for each

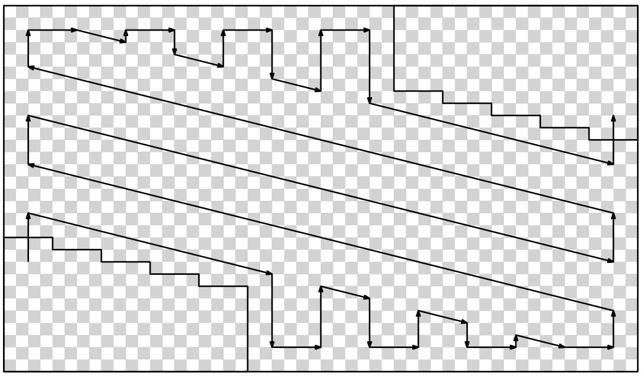


Figure 20 The formation moves of a giraffe's tour on a 52 × 30 board.

giraffe (Figure 18 shows one in red). Therefore, $T_{w,h}$ is equivalent to U $2(l+k)$ times in a row. Note that after eight consecutive upward moves, or U twice, each giraffe ends where it started. Thus, this is true of the entire tour.

6 Conclusions

We have introduced two new metrics of "simplicity" for knight's tours: the number of turns and the number of crossings. We provided an algorithm which is within a constant factor of the minimum for both metrics. In doing so, we found that, in a $n \times n$ board, the minimum number of turns and crossings is $O(n)$. Prior techniques such as divide-and-conquer necessarily result in $\Theta(n^2)$ turns and crossings, so at the outset of this work it was unclear whether $o(n^2)$ could be achieved at all.

The ideas of the algorithm, while simple, seem to be new in the literature, which is interesting considering the history of the problem. Perhaps it was our *a priori* optimization goal that led us in a new direction. The algorithm exhibits a number of positive traits. It is simple, efficient to compute, parallelizable, and amenable to generalizations (see Section 4). We conclude with some open questions:

- Our tours have $9.5n + O(1)$ turns and $13n + O(1)$ crossings, and we showed respective lower bounds of $(6 - \varepsilon)n$ and $4n - O(1)$. The main open question is closing or reducing these gaps, as there may still be room for improvement in both directions. We conjecture that the minimum number of turns is at least $8n$.
- Are there other properties of knight's tours, besides turns and crossings, that might be interesting to optimize?
- Our method relies heavily on the topology of the knight-move graph. Thus, it is not applicable to general Hamiltonian cycle problems. Are there other graph classes with a similar enough structure that the ideas of formations and junctions can be useful?

References

1 Karla Alwan and Kelly Waters. Finding re-entrant knight's tours on n-by-m boards. In *Proceedings of the 30th Annual Southeast Regional Conference*, ACM-SE 30, pages 377–382, New York, NY, USA, 1992. ACM. doi:10.1145/503720.503806.

2 W.W. Rouse Ball and H.S.M. Coxeter. *Mathmatical Recreations & Essays: 12th Edition*. University of Toronto Press, 1974. URL: http://www.jstor.org/stable/10.3138/j.ctt15jjcrn.

3 John D. Beasley. Magic knight's tours. *The College Mathematics Journal*, 43(1):72–75, 2012. URL: http://www.jstor.org/stable/10.4169/college.math.j.43.1.072.

4 Ernest Bergholt. Three memoirs on knight's tours. *The Games and Puzzles Journal*, 2(18):327–341, 2001.

5 G.L. Chia and Siew-Hui Ong. Generalized knight's tours on rectangular chessboards. *Discrete Applied Mathematics*, 150(1):80–98, 2005. doi:10.1016/j.dam.2004.11.008.

6 Axel Conrad, Tanja Hindrichs, Hussein Morsy, and Ingo Wegener. Solution of the knight's hamiltonian path problem on chessboards. *Discrete Applied Mathematics*, 50(2):125–134, 1994. doi:10.1016/0166-218X(92)00170-Q.

7 Paul Cull and Jeffery De Curtins. Knight's tour revisited. *Fibonacci Quarterly*, 16:276–285, June 1978.

8 Italo J. Dejter. Equivalent conditions for euler's problem on z_4-hamilton cycles. *Ars Combinatoria*, 16–B:285–295, 1983.

9 Joe DeMaio. Which chessboards have a closed knight's tour within the cube? *the electronic journal of combinatorics*, 14(1):32, 2007.

10 Joe DeMaio and Mathew Bindia. Which chessboards have a closed knight's tour within the rectangular prism? *the electronic journal of combinatorics*, 18(1):14, 2011.

11 Joshua Erde, Bruno Golénia, and Sylvain Golénia. The closed knight tour problem in higher dimensions. *the electronic journal of combinatorics*, 19(4):9, 2012.

12 Alexander Fischer. New records in nonintersecting knight paths. *The Games and Puzzles Journal*, 2006.

13 Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

14 George P. Jelliss. Non-intersecting paths by leapers. *The Games and Puzzles Journal*, 2(17):305–310, 1999.

15 George P. Jelliss. Symmetry in knight's tours. *The Games and Puzzles Journal*, 2(16):282–287, 1999.

16 Nina Kamčev. Generalised knight's tours. *the electronic journal of combinatorics*, 21(1):32, 2011.

17 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics*, 23(3):309–311, 1978.

18 Donald E. Knuth. Leaper graphs. *The Mathematical Gazette*, 78(483):274–297, 1994. URL: `http://www.jstor.org/stable/3620202`.

19 Awani Kumar. Non-crossing Knight's Tour in 3-Dimension. *ArXiv e-prints*, March 2008. `arXiv:0803.4259`.

20 Olaf Kyek, Ian Parberry, and Ingo Wegener. Bounds on the number of knight's tours. *Discrete Applied Mathematics*, 74(2):171–181, 1997. `doi:10.1016/S0166-218X(96)00031-5`.

21 Shun-Shii Lin and Chung-Liang Wei. Optimal algorithms for constructing knight's tours on arbitrary $n \times m$ chessboards. *Discrete Applied Mathematics*, 146(3):219–232, 2005. `doi:10.1016/j.dam.2004.11.002`.

22 Stephen R. Mahaney. Sparse complete sets for np: Solution of a conjecture of berman and hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982. `doi:10.1016/0022-0000(82)90002-2`.

23 Brendan D. McKay. Knight's tours of an 8× 8 chessboard. Technical report, Australian National University, Department of Computer Science, February 1997.

24 Crispin Nash-Williams. Abelian groups, graphs and generalized knights. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 55(3), pages 232–238. Cambridge University Press, 1959.

25 Ian Parberry. Scalability of a neural network for the knight's tour problem. *Neurocomputing*, 12(1):19–33, 1996. `doi:10.1016/0925-2312(95)00027-5`.

26 Ian Parberry. An efficient algorithm for the knight's tour problem. *Discrete Applied Mathematics*, 73(3):251–260, 1997.

27 Ira Pohl. A method for finding hamilton paths and knight's tours. *Commun. ACM*, 10(7):446–449, July 1967. `doi:10.1145/363427.363463`.

28 Yulan Qing and John J. Watkins. Knight's tours for cubes and boxes. *Congressus Numerantium*, January 2006.

29 Allen J. Schwenk. Which rectangular chessboards have a knight's tour? *Mathematics Magazine*, 64(5):325–332, 1991.

30 Jefferey A. Shufelt and Hans J. Berliner. Generating knight's tours without backtracking from errors. Technical report, Carnegie-Mellon University, School of Computer Science, 1993.

31 Douglas Squirrel and Paul Cull. A warnsdorff-rule algorithm for knight's tours on square chessboards, 1996.

32 John J. Watkins. *Across the Board: The Mathematics of Chessboard Problems*. Princeton Puzzlers. Princeton University Press; Reissue edition, 2012.

33 John J. Watkins and Rebecca L. Hoenigman. Knight's tours on a torus. *Mathematics Magazine*, 70(3):175–184, 1997. `doi:10.1080/0025570X.1997.11996528`.

34 L. D. Yarbrough. Uncrossed knight's tours. *Journal of Recreational Mathematics*, 1(3):140–142, 1969.

A Computational Complexity

Consider the following decision versions of the problems: is there a knight's tour on an $n \times n$ board with at most k turns (resp. crossings)? We do not know if these problems are in P. Furthermore, it may depend on how the input is encoded. Technically, the input consists of two numbers, n and k, which can be encoded in $O(\log n + \log k)$ bits. However, it is more natural to do the analysis as a function of the board size (or, equivalently, of the underlying graph on which we are solving the Hamiltonian Cycle problem), that is, $\Theta(n^2)$. It is plausible that the optimal number of turns (resp. crossings) is a simple, arithmetic function of n. This would be the case if the optimal tour follows a predictable pattern like our construction (counting the number of turns or crossings achieved by our algorithm does not require finding the tour). In this case, the problems are in P, regardless of the input's encoding.

If the input is represented using $\Theta(n^2)$ space, the problems are clearly in NP, as a tour with k turns/crossings acts as a certificate of polynomial length. However, unless $\mathsf{P} = \mathsf{NP}$, the problems are not NP-hard.

Consider the language $\{1^n 0 1^k \mid \text{ a tour exists with at most } k \text{ turns in an } n \times n \text{ board}\}$, and analogously for crossings. These languages are sparse, meaning that, for any given word length, there is a polynomial number of words of that length in the language. Mahaney's theorem states that if a sparse language is NP-complete, then $\mathsf{P} = \mathsf{NP}$ [22]. This suggests that the problems are in P, though technically they could also be NP-intermediate.

If the input is represented using $O(\log n + \log k)$ bits, then the problems are in NEXP because the "unary" versions above are in NP. Note that, in this setting, simply listing a tour would require time exponential on the input size.

B Easy Lower Bound for Turns

A sketch for a simple proof providing a loose lower bound on the number of turns.

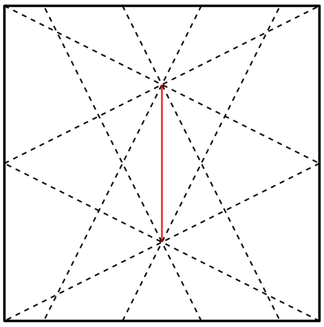


Figure 21 A lower bound of $4.25n - O(1)$ on the number of turns required by any knight's tour on a $n \times n$ board can be seen as follows. Consider the cells in one of the two central columns (does not matter which one), and in a row in the range $(n/4, 3n/4)$. They are shown in red. These cells have the property that every maximal sequence of knight moves without turns through them reaches opposite facing edges. The maximal sequences of knight moves through the first and last red cells are shown in dashed lines. Because n must be even, one of the two endpoints of each maximal sequence through a red cell is not an edge cell. It follows that each red cell is part of a sequence of knight's moves that ends in a turn at a non-edge cell. Thus, there is at least one turn at a non-edge cell for each pair of red cells. Since there are $0.5n$ red cells, we get the mentioned lower bound.

Cutting Bamboo down to Size

Davide Bilò
Department of Humanities and Social Sciences, University of Sassari, Italy
davide.bilo@uniss.it

Luciano Gualà
Department of Enterprise Engineering, University of Rome "Tor Vergata", Italy
guala@mat.uniroma2.it

Stefano Leucci
Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Italy
stefano.leucci@univaq.it

Guido Proietti
Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Italy
Institute for System Analysis and Computer Science "Antonio Ruberti" (IASI CNR), Rome, Italy
guido.proietti@univaq.it

Giacomo Scornavacca
Department of Humanities and Social Sciences, University of Sassari, Italy
giacomo.scornavacca@graduate.univaq.it

Abstract

This paper studies the problem of programming a robotic panda gardener to keep a bamboo garden from obstructing the view of the lake by your house.

The garden consists of n bamboo stalks with known daily growth rates and the gardener can cut at most one bamboo per day. As a computer scientist, you found out that this problem has already been formalized in [Gąsieniec et al., SOFSEM'17] as the *Bamboo Garden Trimming (BGT) problem*, where the goal is that of computing a perpetual schedule (i.e., the sequence of bamboos to cut) for the robotic gardener to follow in order to minimize the *makespan*, i.e., the maximum height ever reached by a bamboo.

Two natural strategies are `Reduce-Max` and `Reduce-Fastest`(x). `Reduce-Max` trims the tallest bamboo of the day, while `Reduce-Fastest`(x) trims the fastest growing bamboo among the ones that are taller than x. It is known that `Reduce-Max` and `Reduce-Fastest`(x) achieve a makespan of $O(\log n)$ and 4 for the best choice of $x = 2$, respectively. We prove the first constant upper bound of 9 for `Reduce-Max` and improve the one for `Reduce-Fastest`(x) to $\frac{3+\sqrt{5}}{2} < 2.62$ for $x = 1 + \frac{1}{\sqrt{5}}$.

Another critical aspect stems from the fact that your robotic gardener has a limited amount of processing power and memory. It is then important for the algorithm to be able to *quickly* determine the next bamboo to cut while requiring at most linear space. We formalize this aspect as the problem of designing a *Trimming Oracle* data structure, and we provide three efficient Trimming Oracles implementing different perpetual schedules, including those produced by `Reduce-Max` and `Reduce-Fastest(x)`.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases bamboo garden trimming, trimming oracles, approximation algorithms, pinwheel scheduling

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.5

Supplementary Material An interactive implementation of the Trimming Oracles described in this paper is available at `https://www.isnphard.com/g/bamboo-garden-trimming/`.

Funding *Davide Bilò*: This work was partially supported by the Research Grant FBS2016_BILO, funded by "Fondazione di Sardegna" in 2016.
Giacomo Scornavacca: This work was partially supported by Research Grant FBS2016_BILO, funded by "Fondazione di Sardegna" in 2016.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 5; pp. 5:1–5:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements The authors would like to thank Francesca Marmigi for the picture of the robotic panda gardener in Figure 1. We are also grateful to an anonymous reviewer whose comments allowed us to significantly simplify the analysis of `Reduce-Max`.

1 Introduction

You just bought a house by a lake. A bamboo garden grows outside the house and obstructs the beautiful view of the lake. To solve the problem, you also bought a robotic panda gardener which, once per day, can instantaneously trim a single bamboo. You have already measured the growth rate of every bamboo in the garden, and you are now faced with programming the gardener with a suitable schedule of bamboos to trim in order to keep the view as clear as possible.

This problem is known as the *Bamboo Garden Trimming (BGT) Problem* [11] and can be formalized as follows: the garden contains n bamboos $b_1, \dots, b_n$, where bamboo b_i has a known daily growth rate of $h_i > 0$, with $h_1 \geq \ldots \geq h_n$ and $\sum_{i=1}^{n} h_i = 1$. Initially, the height of each bamboo is 0, and at the end of each day, the robotic gardener can trim at most one bamboo to instantaneously reset its height to zero. The height of bamboo b_i at the end of day $d \geq 1$ and before the gardener decides which bamboo to trim is equal to $(d - d')h_i$, where $d' < d$ is the last day preceding d in which b_i was trimmed (if b_i was never trimmed before day d, then $d' = 0$). See Figure 1 for an example.

The main task in BGT is to design a perpetual trimming schedule that keeps the tallest bamboo ever seen in the garden as short as possible. In the literature of scheduling problems, this maximum height is called *makespan*.

A simple observation shows that the makespan must be at least 1 for every instance. Indeed, for any $\epsilon > 0$, a makespan of $1 - \epsilon$ would imply that the daily amount of bamboo cut from the garden is at most $1 - \epsilon$, while the overall daily growth rate of the garden is 1. This is a contradiction. Furthermore, there are instances for which the makespan can be made arbitrarily close to 2. Consider, for example, two bamboos b_1, b_2 with daily growth rates $h_1 = 1 - \epsilon$ and $h_2 = \epsilon$, respectively. Clearly, when bamboo b_2 must be cut, the height of b_1 becomes at least $2 - 2\epsilon$. This implies that the best makespan one can hope for is 2.

Two natural strategies are known for the BGT problem, namely `Reduce-Max` and `Reduce-Fastest`(x). The former consists of trimming the tallest bamboo at the end of every day, while the latter cuts the bamboo with fastest growth rate among those having a height of at least x. Experimental results show that `Reduce-Max` performs very well in practice as it seems to guarantee a makespan of 2 [2, 10]. However, the best known upper bound to the makespan is $1 + \mathcal{H}_{n-1} = \Theta(\log n)$, where $\mathcal{H}_{n-1}$ is the $(n-1)$-th harmonic number [5]. Interestingly, this $\Theta(\log n)$ bound also holds for the adversarial setting in which at every day an adversary decides how to distribute the unit of growth among all the bamboos. In this adversarial case such upper bound can be shown to be tight, while understanding whether `Reduce-Max` achieves a constant makespan in the non-adversarial setting is a major open problem [11, 10]. On the other hand, in [11] it is shown that `Reduce-Fastest`(x) guarantees a makespan of 4 for $x = 2$. Furthermore, it is also conjectured that `Reduce-Fastest`(1) guarantees a makespan of 2 [10].

In [11], the authors also provide a different algorithm guaranteeing a makespan of 2. This is obtained by transforming the BGT problem instance into an instance of a related scheduling problem called *Pinwheel Scheduling*, by suitably rounding the growth rates of the bamboos. Then, a perpetual schedule for the Pinwheel Scheduling instance is computed

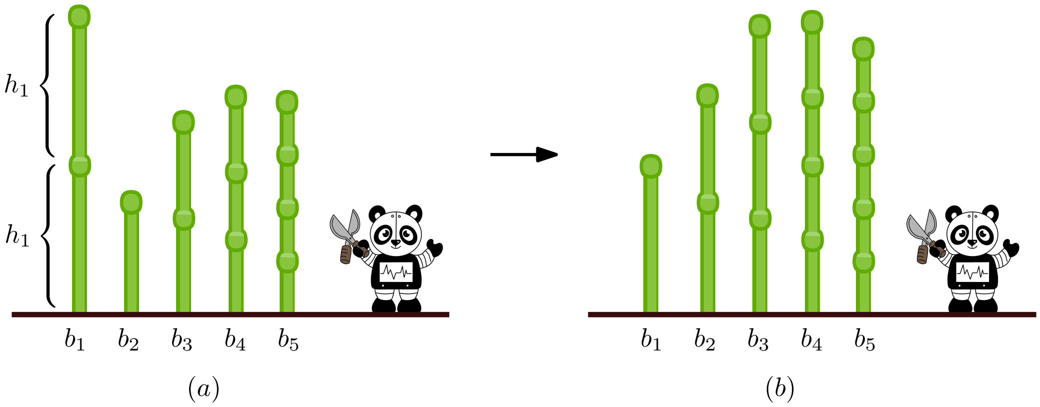


Figure 1 (a) The bamboo garden at the end of a day, just before the robotic gardener trims bamboo b_1. (b) The bamboo garden at the end of the next day, before cutting a bamboo.

using existing algorithms [8, 13]. It turns out that this approach has a problematic aspect since it is known that *any* perpetual schedule for the Pinwheel Scheduling instance can have length $\Omega\left(\prod_{i=1}^{n}\frac{1}{h_i}\right)$ in the worst case.

The above observation gives rise to the following complexity issue: Can a perpetual schedule be efficiently implemented in general? Essentially, a solution consists of designing a *trimming oracle*, namely a *compact* data structure that is able to *quickly* answer to the query "What is the next bamboo to trim?".

It is worth noticing that similar problems are discussed in [11], where the authors ask for the design of trimming oracles that implement known BGT algorithms. For example, they explicitly leave open the problem of designing an oracle implementing `Reduce-Max` with query time of $o(n)$.

Our results. Our contribution is twofold. In Section 2, we provide the following improved analyses of `Reduce-Max` and `Reduce-Fastest`(x):

- We show that `Reduce-Max` achieves a makespan of at most 9. This is the first constant upper bound for this strategy and shows a separation between the static and the adversarial setting for which the makespan is known to be $\Theta(\log n)$.
- We show that, for any $x > 1$, `Reduce-Fastest`(x) guarantees a makespan of at most $\max\left\{x+\frac{x^2}{4(x-1)}, \frac{1}{2}+x+\frac{x^2}{4(x-\frac{1}{2})}\right\}$. For the best choice of $x = 1+\frac{1}{\sqrt{5}}$, this results in a makespan of $1+\phi = \frac{3+\sqrt{5}}{2} < 2.62$, where ϕ is the golden ratio. Notice also that for $x = 2$ (the best choice of x according to the analysis of [11]) we obtain an upper bound of 19/6 which improves over the previously known upper bound of 4.

Then, in Section 3, we provide the following trimming oracles:

- A trimming oracle implementing `Reduce-Max` whose query time is $O(\log^2 n)$ in the worst-case or $O(\log n)$ amortized. The size of the oracle is $O(n)$ while the time needed to build it is $O(n \log n)$. This answers the open problem given in [11].
- A trimming oracle implementing `Reduce-Fastest`(x) with $O(\log n)$ worst-case query time. This oracle has linear size and can be built in $O(n \log n)$ time.
- A trimming oracle guaranteeing a makespan of 2. This oracle uses the rounding strategy from [11] but it uses a different approach to compute a perpetual schedule. Our oracle answers queries in $O(\log n)$ amortized time, requires $O(n)$ space, and can be built in $O(n \log n)$ time.

This result favorably compares with the existing oracles achieving makespan 2 implicitly obtained when the reduction of [11] is combined with the results in [13, 8] for the Pinwheel Scheduling problem. Indeed, once the instance G of BGT has been transformed into an instance P of Pinwheel Scheduling, any oracle implementing a feasible schedule for P is an oracle for G with makespan 2. In [13], the authors show how to compute a schedule for P of length $L = \Omega(\prod_{i=1}^{n} \frac{1}{h_i})$, which results in an oracle with exponential building time and constant query time. In [8], an oracle having query time of $O(1)$ is claimed, but attaining such a complexity requires the use of $\Theta(n)$ parallel processors and the ability to perform arithmetic operations modulo L (whose binary representation may need $\Omega(n)$ bits) in constant time.

An interactive implementation of our Trimming Oracles described above is available at `https://www.isnphard.com/g/bamboo-garden-trimming/`.

Other related work. The BGT problem has been introduced in [11]. Besides the aforementioned results, this paper also provides an algorithm achieving a makespan better than 2 for a subclass of instances with *balanced* growth rates; informally, an instance is said to be balanced if at least a constant fraction of the overall daily growth is due to bamboos $b_2, \dots, b_n$. The authors also introduce a generalization of the problem, named *Continuous BGT*, where each bamboo b_i grows continuously at a rate of h_i per unit of time and is located in a point of a metric space. The gardener can instantaneously cut a bamboo that lies in its same location, but needs to move from one bamboo to the next at a constant speed. Notice that this is a generalization of BGT problem since one can consider the trivial metric in which all distances are 1 (and it is never convenient for the gardener to remain in the same location).

Another generalization of the BGT problem called *cup game* can be equivalently formulated as follows: each day the gardener can reduce the height of a bamboo by up to $1 + \epsilon$ units, for some constant parameter $\epsilon \geq 0$. If the growth rates can change each day and an adversary distributes the daily unit of growth among the bamboos, then a (tight) makespan of $O(\log n)$ can still be achieved. If the gardener's algorithm is randomized and the adversary is *oblivious*, i.e., it is aware of gardener's algorithm but does not know the random bits or the previously trimmed bamboos, then the makespan is $O(m)$ with probability at least $1 - O(2^{-2^m})$, i.e., it is $O(\log \log n)$ with high probability [4]. The generalization of the cup game with multiple gardeners has been also addressed in [4, 15].

As we already mentioned, a problem closely related to BGT is the Pinwheel Scheduling problem that received a lot of attention in the literature [7, 8, 13, 14, 16, 19].

The BGT problem and its generalizations also appeared in a variety of other applications, ranging from deamortization, to buffer management in network switches, to quality of service in real-time scheduling (see, e.g., [3, 12, 1] and the references therein).

2 New bounds on the makespan of known BGT algorithms

In this section we provide an improved analysis on the makespan guaranteed by the `Reduce-Fastest`(x) strategy and the first analysis that upper bounds the makespan of `Reduce-Max` by a constant. In the rest of this section, we say that a bamboo b_i is trimmed at day d to specify that the schedule computed using the heuristic chooses b_i as the bamboo that has to be trimmed at the end of day d.

2.1 The analysis for Reduce-Max

Here we analyze the heuristic Reduce-Max, that consists in trimming the tallest bamboo at the end of each day (ties are broken arbitrarily).

▶ **Theorem 1.** *Reduce-Max guarantees a makespan of* 9.

Proof. We partition the bamboos into groups, that we call levels, according to their daily growth rates. More precisely, we say that bamboo b_i is of *level* $j \geq 1$ if $\frac{1}{2^j} \leq h_i < \frac{1}{2^{j-1}}$. Let K be the level of bamboo b_n and, for every j, with $1 \leq j \leq K$, let L_j denote the set of all the bamboos of level j.

For every $1 \leq j \leq K$, let $\sigma(j)$ be the maximum height ever reached by any bamboo of level $k \geq j$, with $\sigma(K+1) = 0$ by definition. In order to bound the makespan, it suffices to bound $\sigma(1)$. Rather than doing this directly, we will instead show that for $1 \leq j \leq K$, we have

$$\sigma(j) \leq \max\Big\{3, \sigma(j+1)\Big\} + 3\sum_{k=1}^{j} \frac{|L_k|}{2^j}. \tag{1}$$

Let $q \leq K$ be the level with lowest index such that $\sigma(q) \leq 3$ (if there is no such index, $q = K$). For any $j < q$ it holds $\max\Big\{3, \sigma(j+1)\Big\} = \sigma(j+1)$. As a consequence, the makespan is at most

$$\sigma(1) \leq 3 + \sum_{j=1}^{q} 3 \sum_{k=1}^{j} \frac{|L_k|}{2^j} \leq 3 + 3\sum_{j=1}^{K}\sum_{k=1}^{j} \frac{|L_k|}{2^j}. \tag{2}$$

If bamboo b_i is of level s, then the bamboo stalk contributes $\sum_{j=s}^{K} \frac{1}{2^j} < \frac{2}{2^s} \leq 2h_i$ to the sum in (2). As $\sum_{i=1}^{n} h_i = 1$ by definition, it follows that the makespan is bounded by

$$\sigma(1) \leq 3 + 3\sum_{j=1}^{K}\sum_{k=1}^{j} \frac{|L_k|}{2^j} \leq 3 + 6\sum_{i=1}^{n} h_i = 9.$$

We now complete the proof by proving (1), which compares $\sigma(j)$ and $\sigma(j+1)$ for all j. Suppose that bamboo b_i has level j, and that at the end of day d_1 bamboo b_i achieves the maximum height ever reached by any bamboo of level j. Let $d_0 < d_1$ be the largest-numbered day prior to d_1 at the end of which either (a) a bamboo b_ℓ with level greater than j was trimmed, or (b) a bamboo b_ℓ with height less than 3 was trimmed. Because the Reduce-Max algorithm always trims the tallest bamboo, the height of b_i at the end of day d_0 is at most the height of b_ℓ at the end of day d_0, right before b_ℓ is trimmed. It follows that the height of b_i at the end of day d_1, right before b_i is trimmed, is at most $h_i(d_1 - d_0)$ greater than the height of b_ℓ at the end of day d_0, right before b_ℓ is trimmed. Since the height of b_ℓ at the end of day d_0 is at most $\max\{3, \sigma(j+1)\}$, it follows that

$$\sigma(j) \leq \max\{3, \sigma(j+1)\} + h_i(d_1 - d_0) < \max\{3, \sigma(j+1)\} + \frac{2}{2^j}(d_1 - d_0), \tag{3}$$

where in the last inequality we use the fact that $h_i < \frac{1}{2^{j-1}}$. Now, in order to prove (1), it suffices to show that $d_1 - d_0 \leq \frac{3}{2}\sum_{k=1}^{j} |L_k|$. By the definition of d_0, at any day $t \in [d_0+1, d_1]$ a bamboo of height at least 3 and with level equal or smaller than j is trimmed. We call a cut at day $t \in [d_0+1, d_1]$ a *repeated cut* if, at day t, a bamboo that was already trimmed at any day in $[d_0+1, t-1]$ is trimmed again, and a *first cut* otherwise. Note that each repeated cut trims a bamboo whose growth occurred entirely during days $[d_0+1, t-1]$ and that the total growth of the forest in the interval interval $[d_0+1, d_1]$ is $d_1 - d_0$. It means that at most $\frac{1}{3}$ of the cuts at day $t \in [d_0+1, d_1]$ can be repeated cuts, since at the end of each of these days a bamboo of height at least 3 is trimmed. On the other hand, the number of first cuts is bounded by the number of distinct bamboos with levels less or equal to j, i.e., by $\sum_{k=1}^{j} |L_k|$. It follows that the number of days in the window $[d_0+1, d_1]$ satisfies $d_1 - d_0 \leq \frac{1}{3}(d_1 - d_0) + \sum_{k=1}^{j} |L_k|$, and thus $d_1 - d_0 \leq \frac{3}{2} \sum_{k=1}^{j} |L_k|$ as desired. ◀

2.2 The analysis for Reduce-Fastest(x)

Here we provide an improved analysis of the makespan achieved by the Reduce-Fastest(x) strategy. The heuristic Reduce-Fastest(x) consists in trimming, at the end of each day, the bamboo with the fastest daily growth rate among those that have reached a height of at least x (ties are broken in favour of the bamboo with the smallest index).

▶ **Theorem 2.** *The makespan of Reduce-Fastest(x), for a constant x such that $x > 1$, is upper bounded by* $\max\left\{x + \frac{x^2}{4(x-1)}, \frac{1}{2} + x + \frac{x^2}{4(x-\frac{1}{2})}\right\}$.

Proof. Let M be the makespan of Reduce-Fastest(x) and let b_i be one of the bamboos such that the maximum height reached by b_i is exactly M. Let $[d_0, d_1]$ be an interval of days such that b_i reaches the makespan in d_1 and d_0 is the last day in which b_i was trimmed before d_1 (d_0 may also be equal to 0). Let δ the first day in $[d_0, d_1]$ such that the height of b_i is at least x. For sake of simplicity we rename the interval $[\delta, d_1]$ as $[0, T]$, with $T = d_1 - \delta$. Let N be the number of distinct bamboos that are trimmed in $[0, T-1]$.

We now give some definitions. Let the *volume* V of the garden be the overall growth of the bamboo in the days of the interval $[0, T-1]$. Since the garden grows by $\sum_{i=1}^{n} h_i = 1$ per day, we have $V = T$. Consider the cut of a bamboo b_j on day $d \in [0, T-1]$. If b_j was cut at least once in $[0, d-1]$ we say that the cut is a *repeated cut* otherwise we will say that the cut is a *first cut*. The act of cutting bamboo b_j on a day $d \in [0, T-1]$ with a repeated cut *removes* an amount of volume that is equal to $(d - d')h_j$, where d' is the last day of $[0, d-1]$ in which b_j has been cut, if this is a repeated cut, and $d' = 0$ if this is a first cut. Finally, the *leftover volume* of a bamboo b_j is the overall growth of b_j that happened during interval $[0, T-1]$ and has not been cut by the end of day $T-1$.

We will now bound the amount V' of volume V that is removed by repeated cuts in the interval $[0, T-1]$. Notice that, for each bamboo b_j that is cut in the interval $[0, T-1]$, it holds that $h_j \geq h_i$. If b_j is cut for its first time at day d (among the days in $[0, T-1]$), then the removed volume will be at least $(d+1)h_j \geq (d+1)h_i$. Therefore, after all the N bamboos of the interval $[0, T-1]$ have been cut at least once, the amount of volume removed by first cuts will be at least $\sum_{j=i}^{N} jh_i = \frac{N(N+1)}{2} \cdot h_i$, since at most one bamboo is cut per day. Moreover, if b_j is cut for its last time at day $T-1-d$ (among the days in $[0, T-1]$), b_j will have a height of dh_i at the end of day $T-1$. Finally, bamboo h_i is never cut in the interval $[0, T-1]$ and hence during the interval $[0, T-1]$ it grows by exactly Th_i. This means that the overall leftover volume will be at least $\sum_{j=1}^{N}(j-1)h_i + Th_i = \frac{N(N-1)}{2} \cdot h_i + Th_i$.

We can then write

$$V' \leq V - \left(\frac{N(N+1)}{2} + \frac{N(N-1)}{2}\right) \cdot h_i - Th_i = V - N^2 h_i - Th_i = T(1 - h_i) - N^2 h_i,$$

where the last equality follows from $V = T$.

Since in $[0, T-1]$ the bamboo b_i has height at least x, each repeated cut removes at least x units of volume from V'. Therefore, the number N' of repeated cuts is at most

$\frac{V'}{x} \leq \left(T(1-h_i) - N^2 h_i\right)/x$. We now use this upper bound on N' to derive an upper bound to the time T:

$$T = N + N' \leq N + \frac{T(1-h_i) - N^2 h_i}{x}.$$

For $T'(N) = (Nx - N^2 h_i)/(h_i + x - 1)$, the above formula implies $T \leq T'(N)$. If we fix h_i and x, $T'(N)$ is a concave downward parabola that attains its maximum in its vertex at $N = x/2h_i$. Thus:

$$T \leq T'(x/2h_i) \leq \frac{\frac{x^2}{2h_i} - \frac{x^2}{4h_i}}{h_i + x - 1} = \frac{x^2}{4h_i(h_i + x - 1)}.$$

Using this upper bound to T we now bound the overall growth of the bamboo b_i, i.e., the makespan M. At day $d = 0$, b_i has height at most $x + h_i$ by our choice of δ, and in the next T days it grows by Th_i. Hence:

$$M \leq x + h_i + Th_i < x + h_i + \frac{x^2}{4(h_i + x - 1)}. \tag{4}$$

Let $M'(h_i) = x + h_i + \frac{x^2}{4(h_i+x-1)}$. The derivative w.r.t. h_i of the above formula is

$$\frac{\partial M'}{\partial h_i} = 1 - x^2/4(h_i + x - 1)^2 = \frac{4(h_i + x - 1)^2 - x^2}{4(h_i + x - 1)^2} = \frac{(x + 2h_i - 2)(3x + 2h_i - 2)}{4(h_i + x - 1)^2}.$$

The denominator is always positive, and the numerator is a concave upward parabola having its two roots at $h_i = 1 - 3x/2$ and at $h_i = 1 - x/2$. Let us briefly restrict ourselves to the case $h_i \leq \frac{1}{2}$ and notice that, since $x > 1$, the first root is always negative, while the second root is always smaller than $\frac{1}{2}$. It follows that the maximum of $M'(h_i)$ is attained either at $h_i = 0$ or at $h_i = \frac{1}{2}$. Substituting in Equation 4 we get:

$$M \leq \max\left\{x + \frac{x^2}{4(x-1)}, x + \frac{1}{2} + \frac{x^2}{4(x - \frac{1}{2})}\right\}$$

As far as the case $h_i > \frac{1}{2}$ is concerned, notice that it implies $i = 1$ (since if $i \geq 2$ we would have the contradiction $\sum_{i=1}^{n} h_i > \frac{1}{2} \cdot i = 1$) and hence bamboo b_1 is trimmed as soon as its height reaches at least x. The makespan M must then be less than $x + h_1 < x + 1$, which is always smaller than $x + \frac{1}{2} + \frac{x^2}{4(x-\frac{1}{2})} > x + \frac{1}{2} + \frac{1}{2}$. ◀

▶ **Corollary 3.** *The makespan of* `Reduce-Fastest`(2) *is at most* $19/6$ *and the makespan of* `Reduce-Fastest`$(1 + \frac{1}{\sqrt{5}})$ *is at most* $1 + \phi < 2.62$*, where* ϕ *is the golden ratio.*

3 Trimming oracles

This section is devoted to the design of trimming oracles. More precisely, we first design two trimming oracles that implement `Reduce-Fastest`(x) and `Reduce-Max`, respectively. The trimming oracle that implements `Reduce-Fastest`(x) has a $O(\log n)$ worst-case query time, uses linear size and can be built in $O(n \log n)$ time. The trimming oracle that implements `Reduce-Max` has a $O(\log^2 n)$ worst-case query time or a $O(\log n)$ amortized query time, uses linear space, and can be built in $O(n \log n)$ time. We conclude this section by designing a novel trimming oracle that guarantees a makespan of 2 and has a $O(\log n)$ amortized query time. The oracle uses linear size and can be built in $O(n \log n)$ time. For technical convenience, in this section we index days starting from 0, so that at the end of day 0 the gardener can already trim the first bamboo.

An interactive implementation of the Trimming Oracles described in this section is available at `https://www.isnphard.com/g/bamboo-garden-trimming/`.

3.1 A Trimming Oracle implementing `Reduce-Fastest`(x)

We now describe our trimming oracle implementing `Reduce-Fastest`(x). The idea is to keep track, for each bamboo b_i, of the next day d_i at which b_i will be at least as tall as x. When a query at a generic day D is performed, we will then return the bamboo b_i with minimum index i among the ones for which $d_i \geq D$.

To this aim we will make use of a *priority search tree* [17] data structure T to dynamically maintain a collection $P = \{(x_1, y_1), (x_2, y_2), \dots\}$ of 2D points with distinct y coordinates in $\{1, \dots, n\}$ under insertions and deletions while supporting the following queries:

MinYInXRange(T, x_0): report the minimum y-coordinate among those of the points (x_i, y_i) for which $x_i \leq x_0$, if any.

GetX(T, y): report the x-coordinate x_i of the (at most one) point (x_i, y_i) for which $y_i = y$, if any.

All of the above operations on T require time $O(\log |P|)$, as long as all coordinates and query parameters fit in $O(1)$ words of memory.[1]

In our case, the points (x_i, y_i) will be the pairs (d_i, i) for $i = 1, \dots, n$. In such a way, a MinYInXRange query with $x_0 = D$ will return exactly the index i of the bamboo b_i to be cut at the end of day D, if any. After cutting b_i, we *update* T to account for the new day at which the height b_i will be at least x, i.e., we replace the old point (d_i, i) with $(D + \lceil x/h_i \rceil, i)$. Unfortunately, since the trimming oracle is ought to be used perpetually, (the representations of) both d_i and D will eventually require more than a constant number of memory words.

We solve this problem by dividing the days into contiguous intervals $I_0, I_1, \dots$ of n days each, where $I_j = [nj, nj+1, \dots, n(j+1)-1]$, and by using two priority search trees T_1 and T_2 that are associated with the current and the next interval, respectively. This allows us to measure days from the start of the current interval I_j, i.e., if $D = nj + \delta \in I_j$, then we only need to keep track of $\delta \in [0, \dots, n-1]$. In place of (d_i, i), we store the point (δ_i, i) in T_1, where $\delta_i = d_i - nj$. In this way, the previous query with $x_0 = D$ will now correspond to a query with $x_0 = \delta$.

Finally, we also ensure that at the end of the generic day $D = nj + \delta$, T_2 contains the point (δ'_i, i) for each $d_i = n(j+1) + \delta'$ and $i = 1, \dots, \delta + 1$. This allows us to swap T_2 for T_1 when interval I_j ends.

Since bamboo b_i reaches height x exactly $\lceil x/h_i \rceil$ days after being cut, it follows that the largest x-coordinate ever stored in T_1 or T_2 is at most $n + x/h_n$ and we can then support MinYInXRange queries in $O(\log(n))$ time (where we are assuming that x, h_n and thus x/h_n fit in a constant number of memory words).

The pseudocode of our trimming oracle is as given in Algorithm 1. The procedure Query() is intended to be run every day. Consider a generic day δ of the current interval I_j. At this time, T_1 correctly encodes all the days at which the bamboos reached, or will reach, height at least x when measured from the starting day of the current interval (i.e., from day nj), and after all the cuts of the previous days have already been performed.[2] The same information concerning bamboos $b_1, \dots, b_\delta$ is replicated in T_2 with respect to the starting time of the next interval (i.e., $(n+1)j$). The procedure Query() accomplishes two tasks: (1) it computes

[1] While this query is not described in [17], it can be easily implemented in $O(\log |P|)$ time using a dictionary and the fact that y-coordinates are distinct.

[2] Actually, if a bamboo b_i reached height x before the beginning of the considered interval, we will store the point $(0, i)$ in place of (δ_i, i) with $\delta_i < 0$. This still encodes the fact that it is possible to trim b_i from the very fist day of the interval and prevents δ_i from becoming arbitrarily small.

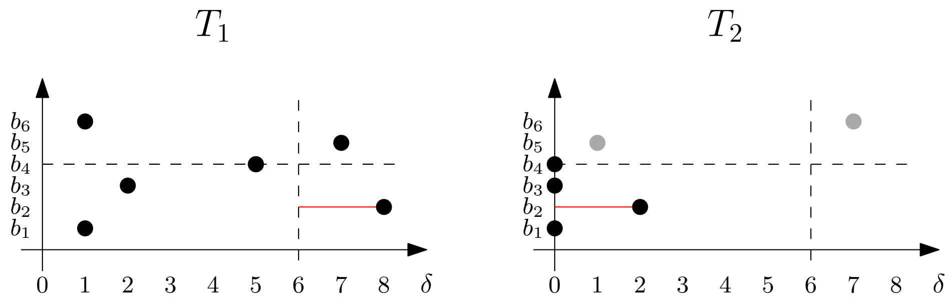


Figure 2 An example of the points contained in the priority search trees T_1 and T_2 for an instance with 6 bamboos at the end of day $\delta = 4$ of a generic interval I_j. We labeled the y-coordinate i with b_i since the unique point (d_i, i) having y-coordinate i represents the day at which b_i reached/will reach a height of at least x. Notice that the points corresponding to bamboos b_1, b_2, b_3, and b_4 are already updated in T_2, while b_5 and b_6 (shown in gray) will be updated by the days $\delta = 5$ and $\delta = 6$, respectively. At the end of day $\delta = 6$, all the points in T_2 are updated and T_1 can be safely swapped with T_2.

the bamboo b_i to cut at the end of day δ of the current interval (if any) and it updates the data structures T_1 and T_2 to account for the new height of b_i; (2) it updates the information concerning $b_{\delta+1}$ in T_2. See Figure 2 for an example.

The following theorem summarizes the performances of our trimming oracle.

▶ **Theorem 4.** *There is a Trimming Oracle implementing* `Reduce-Fastest`(x) *that uses* $O(n)$ *space, can be built in* $O(n \log n)$ *time, and can report the next bamboo to trim in* $O(\log n)$ *worst-case time.*

3.2 A Trimming Oracle implementing `Reduce-Max`

The idea is to maintain collection L of n lines $\ell_1, \ldots, \ell_n$ in which $\ell_i(d) = h_i d + c_i$ is associated with bamboo b_i and represents its height at the end of day d. Initially $c_i = h_i$.

Determining the bamboo b_i to trim at a generic day d then corresponds to finding the index i that maximizes $\ell_i(d)$. After bamboo b_i, previously of height H, has been cut, ℓ_i needs to be updated to reflect the fact that b_i has height 0 at time d, which corresponds to decreasing c_i by H.

The *upper envelope* $\mathcal{U}_L$ of L is a function defined as $\mathcal{U}_L(d) = \max_{\ell \in L} \ell(d)$. We make use of an *upper envelope data structure* U that is able to maintain L under insertions, deletions and lookups of named lines and supports the following query operation:

Upper(U, d) return a line $\ell \in L$ for which $\ell(d) = \mathcal{U}_L(d)$.

Unfortunately, the trivial implementation of the trimming oracle suggested by the above description encounters similar problems as the ones discussed in Section 3.1 for `Reduce-Fastest`(x): the current day d and the coefficients c_i will grow indefinitely, thus affecting the computational complexity.

Once again, we solve this problem by using two copies U_1, U_2 of the previous *upper envelope data structure* and by dividing the days into intervals $I_1, I_2, \ldots$ with $I_j = [nj, nj + 1, \ldots, n(j + 1) - 1]$. At the beginning of the current day $D = nj + \delta \in I_j$, U_1 will contain all lines $\ell_1, \ldots, \ell_n$ and the value of each $\ell_i(\delta)$ will be exactly the height of b_i. Moreover, at the end of day D (i.e., after the highest bamboo of day D has been trimmed), U_2 will contain a line ℓ'_i for each $i \leq \delta + 1$ such that $\ell'_i(\delta')$ with $\delta' \in [0, n - 1]$ is exactly the height reached by b_i on day $n(j + 1) + \delta'$ if it is not trimmed on days $nj + \delta + 1, \ldots, n(j + 1) + \delta' - 1$. This

Algorithm 1 Trimming Oracle for `Reduce-Fastest(x)`.

```
1  Function Build():
2      $\delta \leftarrow 0$;
3      $T_1, T_2 \leftarrow$ Pointers to two empty priority search trees;
4      $h_1, \dots, h_n \leftarrow$ Sort the growth rates of the $n$ bamboo in nonincreasing order;
5      for $i = 1 \dots, n$ do
6          Insert($T_1, (\lceil x/h_i \rceil - 1, i)$)

7  Function Update($T, \delta_i, i$):
8      $\delta'_i \leftarrow$ GetX($T, i$);
9      if $\delta'_i$ exists then Delete(($\delta'_i, i$));
10     Insert($T, (\max\{0, \delta_i\}, i)$);

11 Function Query():
       // Cut fastest bamboo $b_i$ that reached height $x$ by day $\delta$
12     $i \leftarrow$ MinYInXRange($T_1, \delta$);
13     if $i$ exists then
14         Update($T_1$, $\delta + \lceil x/h_i \rceil$, $i$) ;
15         Update($T_2$, $\delta + \lceil x/h_i \rceil - n$, $i$) ;
       // Make sure that bamboo $b_{\delta+1}$ is updated in $T_2$
16     $\delta_{\delta+1} \leftarrow$ GetX($T_1, \delta + 1$);
17     Update($T_2$, $\delta_{\delta+1} - n$, $\delta + 1$)
       // Move to the next day and possibly to the next interval
18     $\delta \leftarrow (\delta + 1) \bmod n$;
19     if $\delta = 0$ then Swap $T_1$ and $T_2$;
20     if $i$ exists then return "Trim bamboo $b_i$" else return "Do nothing";
```

means that at the end of day $nj + (n - 1)$, U_2 correctly describes the heights of all bamboos in the next interval I_{j+1} as a function of δ', and we can safely swap U_1 with U_2. See Figure 3 for an example.

The pseudocode of our trimming oracle is given in Algorithm 2. A technicality concerns the initial construction of the set of lines in U_1. Notice that this is not handled by the Build() function, but we iteratively add $\ell_1, \dots, \ell_n$ during the first n calls to Query() (i.e., during the days of interval I_0). We can safely do this since the `Reduce-Max` strategy ensures that at time $D \in I_0$ only bamboos in $\{b_1, \dots, b_{D+1}\}$ can conceivably be trimmed. This is handled by the test of line 9, which is only true for $D \in I_0$ and will impact our amortized bounds, as noted below.

The performances of our trimming oracle depend on the specific implementation of the upper envelope data structure use. In [18], such a data structure guaranteeing a worst-case time of $O(\log^2 n)$ per operation is given, while a better amortized bound of $O(\log n)$ per operation was obtained in [6].[3] Moreover, from Theorem 1 we know that the makespan of `Reduce-Max` is at most constant, implying that the maximum absolute value of a generic coefficient c_i is at most $O(nh_i) = O(n)$.

[3] Actually, the authors of [18] and [6] design a dynamic data structure to maintain the convex hull of a set of points in the plane. As explained in [6], point-line duality can be used to convert such a structure into one maintaining the upper envelope of a set of linear functions.

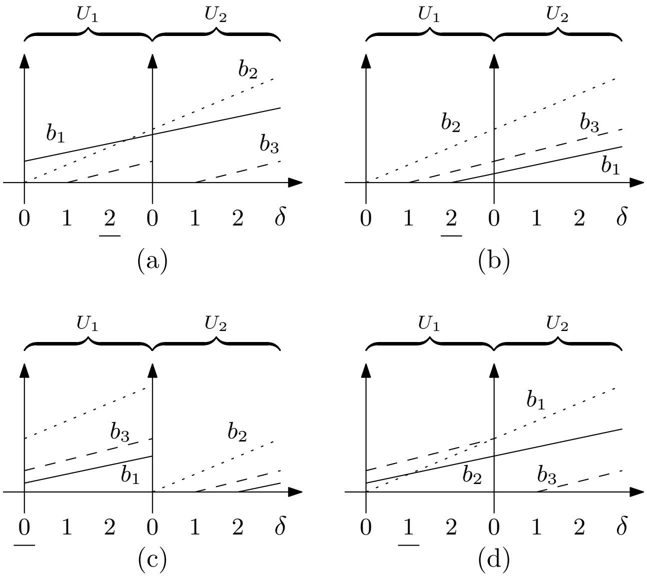


Figure 3 An example of the points contained in U_1 and U_2 for an instance with 3 bamboos, at the beginning of day 2 of a generic interval I_j (a), at the end of day 2 of I_j but before moving to I_{j+1} (b), at beginning of day 0 of I_{j+1} (c), and at the beginning of day 1 of I_{j+1} (d).

The following theorem summarizes the time complexity of our trimming oracle.[4]

▶ **Theorem 5.** *There is a Trimming Oracle implementing* `Reduce-Max` *that uses $O(n)$ space, can be built in $O(n \log n)$ time, and can report the next bamboo to trim in $O(\log^2 n)$ worst-case time, or $O(\log n)$ amortized time.*

3.3 A Trimming Oracle achieving makespan 2

We now design a Trimming Oracle implementing a perpetual schedule that achieves a makespan of at most 2.

We start by rounding the rates $h_1, \dots, h_n$ down to the previous power of $\frac{1}{2}$ as in [11], i.e., we set $h'_i = 2^{\lfloor \log_2 h_i \rfloor}$. We will then provide a perpetual schedule for the rounded instance achieving makespan at most 1 w.r.t. the new rates $h'_1, \dots, h'_2$. Since $h_i \le 2h'_i$, this immediately results in a schedule having makespan at most 2 in the original instance.

Henceforth we assume the input instance is already such that each h_i is a power of $\frac{1}{2}$. Moreover, we will also assume that $\sum_{i=1}^{n} h_i = 1$. Indeed, if $\sum_{i=1}^{n} h_i < 1$ then we can artificially increase some of the growth rates to meet this condition. Clearly, any schedule achieving makespan of most 1 for the transformed instance, also achieves makespan at most 1 in the non-transformed instance.

We transform the instance as follows: we iteratively consider the bamboos in nonincreasing order of rates; when b_i is considered we update h_i to $2^{\left\lfloor \log_2 \left(1 - \sum_{j \ne i} h_j\right) \right\rfloor}$, i.e., to the highest rate that is a power of $\frac{1}{2}$ and still ensures that the sum of the growth rates is at most 1. One can easily check that the above procedure yields an instance for which $\sum_{i=1}^{n} h_i = 1$, as otherwise $\sum_{i=1}^{n} h_i < 1$ and $1 - \sum_{i=1}^{n} h_i \ge h_n$, which is a contradiction since h_n would have been increased to $2h_n$. This requires $O(n \log n)$ time.

[4] Due to lines 9 and 10, the complexity of a query operation is only amortized over the running time of previous queries.

Algorithm 2 Trimming Oracle for `Reduce-Max`.

```
Function Build():
    $\delta \leftarrow 0$;
    $T_1, T_2 \leftarrow$ Pointers to two empty upper envelope data structures;
    $h_1, \ldots, h_n \leftarrow$ Sort the growth rates of the $n$ bamboo in nonincreasing order;
Function Update($U, i, c$):
    Delete($U, \ell_i$);
    Insert($U, \ell_i(d) = h_i d + c$);
Function Query():
    // Ensure that the line $\ell_{\delta+1}$ corresponding to bamboo $b_{\delta+1}$ is in $U_1$
    if there is no line named $\ell_{\delta+1}$ in $U_1$ then
        Insert($U_1, \ell_{\delta+1}(d) = h_{\delta+1} d + h_{\delta+1}$);
    // Cut highest bamboo $b_i$ at day $\delta$
    $\ell_i(d) = h_i d + c_i \leftarrow$ Upper($\delta$);
    Update($U_1, i, -\delta h_i$);
    Update($U_2, i, (n - \delta) h_i$);
    // Ensure that the line $\ell_{\delta+1}$ corresponding to bamboo $b_{\delta+1}$ is updated in $U_2$
    Let $\ell_{\delta+1}(d) = h_{\delta+1} d + c_{\delta+1}$ be the line named $\ell_{\delta+1}$ in $U_1$;
    Update($U_2, \delta + 1, n h_{\delta+1} + c_{\delta+1}$);
    // Move to the next day and possibly to the next interval
    $\delta \leftarrow (\delta + 1) \bmod n$;
    if $\delta = 0$ then Swap $U_1$ and $U_2$;
    return "Trim bamboo $b_i$";
```

In the rest of this section, we will design Trimming Oracles achieving a makespan of at most 1 for instances where all h_is are powers of $\frac{1}{2}$ and $\sum_{i=1}^{n} h_i = 1$.

A Trimming Oracle for regular instances

Let us start by considering an even smaller subset of the former instances, namely the ones in which b_i has a growth rate of $h_i = 2^{-i}$, for $i = 1, \ldots, n-1$, and $h_n = h_{n-1} = 2^{-n+1}$. For the sake of brevity we say that these instances are *regular*.[5]

It turns out that a schedule for regular instances can be easily obtained by exploiting a connection between the index i of bamboo b_i to be cut at a generic day D and the position of the least significant 0 in the last $n-1$ bits in the binary representation of D.

The schedule is as follows: if the last 0 in the binary representation of D appears in the i-th least significant bit, with $i < n$, then b_i is to be cut at the end of day D. Otherwise, if the $n-1$ least significant bits of D are all 1, bamboo b_n is cut at day D.

In this way, the maximum number of days that elapses between any two consecutive cuts of bamboo b_i with $i < n$ is $M_i = 2^i$, while b_n is cut every $M_n = 2^{n-1}$ days. It is then easy to see that, for each bamboo b_i, $h_i \cdot M_i = 1$, thus showing that the resulting makespan is 1 as desired (and this is tight since, in any schedule with bounded makespan, b_1 grows for at least 2 consecutive days). See Figure 4 for an example with $n = 5$.

[5] Notice that, in any regular instance, the grow rates of the bamboos are completely specified by the number n.

D	$(D)_2$	Trim	D	$(D)_2$	Trim	D	$(D)_2$	Trim	D	$(D)_2$	Trim
0	0 0 0 0	b_1	4	0 1 0 0	b_1	8	1 0 0 0	b_1	12	1 1 0 0	b_1
1	0 0 0 1	b_2	5	0 1 0 1	b_2	9	1 0 0 1	b_2	13	1 1 0 1	b_2
2	0 0 1 0	b_1	6	0 1 1 0	b_1	10	1 0 1 0	b_1	14	1 1 1 0	b_1
3	0 0 1 1	b_3	7	0 1 1 1	b_4	11	1 0 1 1	b_3	15	1 1 1 1	b_5

Perpetual schedule: $b_1, b_2, b_1, b_3, b_1, b_2, b_1, b_4, b_1, b_2, b_1, b_3, b_1, b_2, b_1, b_5 \ldots$

Figure 4 A perpetual schedule of a regular instance with 5 bamboos.

This above relation immediately suggests the implementation of a Trimming Oracle that maintains the binary representation of $D \bmod 2^{n-1}$. Since it is well known that a binary counter with n bits can be incremented in $O(1)$ amortized time [9, pp. 454–455], we can state the following:

▶ **Lemma 6.** *For the special case regular instances, there is a Trimming Oracle that uses $O(n)$ space, can be built in $O(n)$ time, can be queried to report the next bamboo to cut in $O(1)$ amortized time, and achieves makespan* 1.

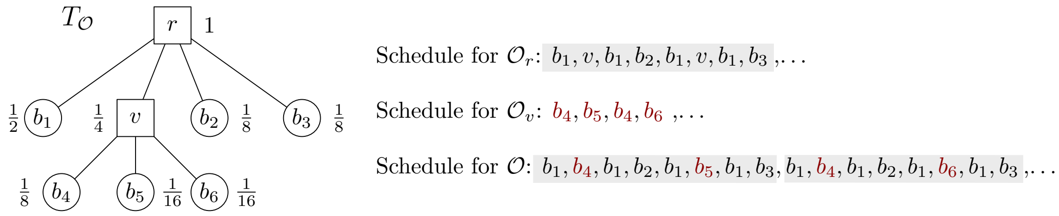
A Trimming Oracle for non-regular instances

Here we show how to design a Trimming Oracle for non-regular instances by iteratively transforming them into suitable regular instances. We will refer to the bamboos $b_1, \ldots, b_n$ as *real bamboos* and will introduce the notion of *virtual bamboos.*

A virtual bamboo v represents a collection of (either real or virtual) bamboos whose growth rates yield a regular instance when suitably scaled by a common factor. The growth rate of v will be equal to the sum of the growth rates of the bamboos in its collection.

To see why this concept is useful, consider an example instance I with 6 bamboos $b_1, \ldots, b_6$ with rates $h_1 = \frac{1}{2}$, $h_2 = \frac{1}{8}$, $h_3 = \frac{1}{8}$, $h_4 = \frac{1}{8}$, $h_5 = \frac{1}{16}$, $h_6 = \frac{1}{16}$. If we replace h_4, h_5, and h_6 with a virtual bamboo v with growth rate $\mathfrak{h} = \frac{1}{8} + \frac{1}{16} + \frac{1}{16} = \frac{1}{4}$ we obtain the related regular instance I' in which the bamboos b_1, v, b_2, b_3 have growth rates $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and $\frac{1}{8}$, respectively. Notice also that the collection of bamboos associated with v is a regular instance I_v once all the rates are multiplied by $\frac{1}{\mathfrak{h}} = 4$. We can now build two Trimming Oracles $\mathcal{O}'$ and $\mathcal{O}_v$ for I' and I_v, respectively, by using Lemma 6. It turns out that $\mathcal{O}'$ and $\mathcal{O}_v$ together allow us to build an oracle $\mathcal{O}_r$ for I as well, which can be represented as a tree (See Figure 5). In general, our oracles $\mathcal{O}$ will consist of a tree $T_{\mathcal{O}}$ whose leaves are the real bamboos $b_1, \ldots, b_n$ of the input instance and in which each internal vertex u serves two purposes: (i) it represents a virtual bamboo whose associated collection C contains the bamboos associated to the children of u; and (ii) it serves as a Trimming Oracle $\mathcal{O}_u$ over the bamboos in C.[6] In order to query $\mathcal{O}$ we proceed as follows: initially we start with a pointer p to the root r of $T_{\mathcal{O}}$; then, we iteratively check whether p points to a leaf ℓ or to an internal vertex u. In the former case, we trim the real bamboo associated with ℓ, otherwise we query the Trimming Oracle $\mathcal{O}_u$ associated with u and we move p to the child of u corresponding to the (virtual or real) bamboo returned by the query on $\mathcal{O}_u$. Since all queries on internal vertices can be performed in $O(1)$ amortized time (see Lemma 6), the amortized time required to query $\mathcal{O}$ is proportional to the height of $T_{\mathcal{O}}$. See Figure 5 for the schedule associated to our example instance I.

[6] The root of $T_{\mathcal{O}}$ can be seen as a virtual bamboo with a growth rate of 1.



Schedule for $\mathcal{O}_r$: $b_1, v, b_1, b_2, b_1, v, b_1, b_3, \dots$

Schedule for $\mathcal{O}_v$: $b_4, b_5, b_4, b_6, \dots$

Schedule for $\mathcal{O}$: $b_1, b_4, b_1, b_2, b_1, b_5, b_1, b_3, b_1, b_4, b_1, b_2, b_1, b_6, b_1, b_3, \dots$

Figure 5 The tree $T_{\mathcal{O}}$ of the Trimming Oracle $\mathcal{O}$ for the instance with 6 bamboos $b_1, \dots, b_6$ with rates $h_1 = \frac{1}{2}$, $h_2 = \frac{1}{8}$, $h_3 = \frac{1}{8}$, $h_4 = \frac{1}{8}$, $h_5 = \frac{1}{16}$, and $h_6 = \frac{1}{16}$. Bamboos b_4, b_5, and b_6 have been replaced by a virtual bamboo v (and a corresponding oracle $\mathcal{O}_v$) with a virtual growth rate of $\frac{1}{4}$. The root r represents both a virtual bamboo with growing rate 1 and the corresponding Trimming Oracle $\mathcal{O}_r$ for the regular instance consisting of b_1, v, b_2, and b_3.

Before showing how to build the tree $T_{\mathcal{O}}$ of our Trimming Oracle $\mathcal{O}$, we prove that the perpetual schedule obtained by querying $\mathcal{O}$ achieves a makespan of at most 1. At any point in time, we say that the *virtual height* of a virtual bamboo v representing a collection C of (real or virtual) bamboos is the maximum over the (real or virtual) heights of the bamboos in C. The bound on the makespan follows by instantiating the following Lemma with $b = r$ and $h = 1$, and by noticing that: (i) the root r of $T_{\mathcal{O}}$ is scheduled every day, and (ii) that the maximum virtual height of r is exactly the makespan.

▶ **Lemma 7.** *Let b be a (real or virtual) bamboo with growth rate h. If b is scheduled at least once every $\frac{1}{h}$ days, then the maximum (real or virtual) height ever reached by b will be at most* 1.

Proof. The proof is by induction on the number η of nodes in the subtree rooted at the vertex representing b in $T_{\mathcal{O}}$.

If $\eta = 1$ then b is a real bamboo and the claim is trivially true since the maximum height reached by b can be at most $h \cdot \frac{1}{h} = 1$.

Suppose then that $\eta \geq 2$ and that the claim holds up to $\eta - 1$. Bamboo b must be a virtual bamboo representing some set $C = \{b'_1, b'_2, \dots, b'_k\}$ of (real or virtual) bamboos which appear as children of b in $T_{\mathcal{O}}$ and are such that: (i) for $i = 1, \dots, k-1$, b'_i has a growth rate of $h'_i = h/2^i$, and (ii) b'_k has a growth rate of $h'_k = h/2^{k-1}$.

Virtual bamboo b schedules the bamboos in C by using the oracle $\mathcal{O}_v$ of Lemma 6, on the regular instance obtained by changing the rate of bamboo b'_i from h'_i to $h''_i = h'_i/h$.

Let d_i (resp. d'_i) be the maximum number of days between any two consecutive cuts of bamboo b'_i according to to the schedule produced by $\mathcal{O}$ (resp. $\mathcal{O}_v$). We know that $d'_i \cdot h''_i \leq 1$ (as otherwise the schedule of $\mathcal{O}_v$ would result in makespan larger than 1 on a regular instance, contradicting Lemma 6), i.e., $d'_i \leq \frac{1}{h''_i}$. Since, b is scheduled at least every $1/h$ days by hypothesis, we have that $d_i \leq \frac{1}{h \cdot h''_i} = \frac{h}{h \cdot h'_i} = \frac{1}{h'_i}$ and hence, by inductive hypothesis, the maximum height reached by b'_i will be at most 1. ◀

We now describe an algorithm that constructs a tree $T_{\mathcal{O}}$ of logarithmic height.

The algorithm employs a collection of sets $S_0, S_1, \dots,$ where initially $S_0 = \{b_1, \dots, b_n\}$ contains all the real bamboos of our input instance, and S_i with $i > 0$ is obtained from S_{i-1} by performing suitable merge operations over the bamboos in S_{i-1}. A merge operation on a collection $C \subseteq S_{i-1}$ of bamboos, whose growth rates yield a regular instance when multiplied by some common factor, consists of: updating S_{i-1} to $S_{i-1} \setminus C$, *creating* a new virtual bamboo v representing C, and adding v to S_i.

The algorithm works in phases. At the generic phase $i = 1, 2, \dots$, it iteratively: (1) looks for a bamboo b with the largest growth rate that can be involved in a merge operation and (2) perform a merge operation on a maximal set $C \subseteq S_{i-1}$ among the ones that contain b (and on which a merge operation can be performed). The procedure is then repeated from step (1) until no suitable bamboo b exists anymore. At this point we name R_{i-1} the current set S_{i-1}, we add to S_i all the bamboos in R_{i-1}, and we proceed to the next phase. The algorithm terminates whenever the set S_i constructed at the end of a phase contains a single virtual bamboo r (of rate 1).

The sequence of merge operations implicitly defines a bottom-up construction of the tree $T_{\mathcal{O}}$, where every merge operation creates a new internal vertex associated with its corresponding virtual bamboo. The root of $T_{\mathcal{O}}$ is r and the height of $T_{\mathcal{O}}$ coincides with the number of phases of the algorithm.

▶ **Lemma 8.** *The algorithm terminates after at most $O(\log n)$ phases.*

Proof. We first prove that the algorithm must eventually terminate. This is a direct consequence of the fact that, at the beginning of any phase i, every set S_{i-1} containing 2 or more bamboos, admits at least one merge operation. Indeed, since merge operations preserve the sum of the growth rates, the overall sum of the rates of the bamboos in S_{i-1} must be 1. Consider now a bamboo $b \in S_{i-1}$ having the lowest growth rate h. Since all rates are powers of $\frac{1}{2}$ and must sum to 1, there must be at least one other bamboo $b' \in S_{i-1} \setminus \{b\}$ having rate h, implying that merge operation can be performed on $C = \{b, b'\}$.

It remains to bound the number of phases. We prove by induction on i that any internal vertex/virtual bamboo v of $T_{\mathcal{O}}$ created at phase i has at least 2^i leaves as descendants. The base case $i = 1$ is trivial since the merge operation that created v must have involved at least 2 real bamboos.

Consider now the case $i \geq 2$. We will show that v was created by a merge operation on a collection C containing at least 2 bamboos v', v'' that were, in turn, created during phase $i - 1$. Hence, by inductive hypothesis, the number of leaves that are descendants of v is the sum of the number of leaves that are descendants of v' and v'', respectively, i.e., it is at least $2^{i-1} + 2^{i-1} = 2^i$.

Let $C \subseteq S_{i-1}$ be the set of bamboos used in the merge operation that created v, and let h be the smallest growth rate among the ones of the bamboos in C. Notice that, by definition of merge operation, there must be 2 distinct bamboos v', v'' with rate h in C. We will now show that v' and v'' were created during phase $i - 1$. We proceed by contradiction. If neither of v' and v'' were created in phase of $i - 1$, then $\{v', v''\} \subseteq R_{i-2}$ which is impossible since $\{v', v''\}$ would have been a feasible merge operation in phase $i - 2$. Assume then that v' was not created in phase $i - 1$, while v'' was created in phase $i - 1$, w.l.o.g. Then, $v' \in R_{i-2}$, while v'' was obtained from a merge operation on a set $C' \subseteq S_{i-2}$ performed in phase $i - 1$. Since the growth rate of v'' is h, the fastest growth rate among the ones of the bamboos in C' must be $h/2$. Hence, the set $C'' = \{v'\} \cup C'$ was a feasible merge operation in phase $i - 1$ when v'' was created. This is a contradiction since $C' \subset C''$ was not a maximal set, as required by the algorithm. ◀

Next Lemma bounds the computational complexity of constructing our oracle.

▶ **Lemma 9.** *The Trimming Oracle $\mathcal{O}$ can be built in $O(n \log n)$ time.*

Proof. It suffices to prove that every phase i of our algorithm can be implemented in $O(n)$ time, since from Lemma 8 the number of phases is $O(\log n)$.

We maintain the set S_{i-1} as a doubly linked list L_{i-1} in which each node ℓ is associated with a distinct growth rate $\mathfrak{h}_\ell$ attained by at least one bamboo in S_{i-1} and stores the set $H(\ell)$ of bamboos of S_{i-1} with grow rate $\mathfrak{h}_\ell$. Nodes appear in decreasing order of $\mathfrak{h}_\ell$. The very first list L_0 can be constructed in $O(n \log n)$ time by sorting the growth rates of the bamboos in S_0. We now show how to build L_i in $O(n)$ time.

The idea is to iteratively find two nodes ℓ_1, ℓ_2 of L_{i-1} such that: (i) ℓ_2 is not the head of L_{i-1} and appears not earlier than ℓ_1; (ii) if ℓ_1 is not the head of L_{i-1}, then selecting one bamboo from the set $H(\ell)$ of each node ℓ that appears before the predecessor ℓ'_1 of ℓ_1, and two bamboos from the set $H(\ell'_1)$ yields the (maximal) set C corresponding the merge operation that algorithm performs; and (iii) all the bamboos in the sets $H(\ell)$ of the nodes ℓ that appear not earlier than ℓ_1 and before ℓ_2 in L_{i-1} will not participate in any merge operation of phase i. We call the set of these bamboos D (notice that it is possible for ℓ_2 to be equal to ℓ_1, in which case no such node ℓ exists and $D = \emptyset$).

To find ℓ_1 and ℓ_2 notice that ℓ_2 is the the last node of L_{i-1} for which any two consecutive nodes preceding ℓ_2 correspond to consecutive rates[7], while the predecessor ℓ'_1 of ℓ_1 is the last node that appears before ℓ_2 and such that $|H(\ell'_1)| \geq 2$.

We now delete the bamboos in $C \cup D$ from their respective sets $H(\ell)$ of L_{i-1}, create a new virtual bamboo v by a merge operation on C. Finally, delete from L_{i-1} all nodes ℓ whose set $H(\ell)$ is now empty. We then repeat this procedure from the beginning until L_{i-1} is empty.

Concerning the time complexity, notice that finding ℓ_1 and ℓ_2 requires $O(k)$ time, where k is the number of nodes that precede ℓ_2 in L_{i-1}. Moreover, all the other steps can be implemented in $O(k)$ time. Therefore, we are able delete k bamboos from L_{i-1} in $O(k)$ time, and hence the overall time complexity to delete all bamboos in L_{i-1} is $O(n)$.

Finally, by keeping track of the sets D, of all the virtual bamboos v generated during the iterations, and by using the fact that the rates of the virtual bamboos are monotonically decreasing, it is also possible to build L_i in $O(n)$ time. ◀

▶ **Lemma 10.** *The Trimming Oracle $\mathcal{O}$ uses $O(n)$ space.*

Proof. By Lemma 6 each internal vertex of $T_{\mathcal{O}}$ maintains a Trimming Oracle with size proportional to the number of its children, implying that the overall space required by $\mathcal{O}$ is proportional to the number η of vertices of $T_{\mathcal{O}}$. Since every internal vertex in $T_{\mathcal{O}}$ has at least 2 children, we have that $\eta = O(n)$. ◀

By combing Lemma 7, Lemma 8, Lemma 9, and Lemma 10, we can state the following theorem that summarizes the result of this section:

▶ **Theorem 11.** *There is a Trimming Oracle that achieves makespan 2, uses $O(n)$ space, can be built in $O(n \log n)$ time, and can report the next bamboo to trim in $O(\log n)$ amortized time.*

References

1 Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, and Mike Paterson. A proportionate fair scheduling rule with good worst-case performance. In Arnold L. Rosenberg and Friedhelm Meyer auf der Heide, editors, *SPAA 2003: Proceedings of*

[7] For technical simplicity, when all consecutive nodes of L_{i-1} correspond to consecutive rates we allow ℓ_1 and/or ℓ_2 to point one position past the end of L_{i-1}.

the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003), pages 101–108. ACM, 2003. doi:10.1145/777412.777430.

2 Sultan S. Alshamrani, Dariusz R. Kowalski, and Leszek Antoni Gasieniec. Efficient discovery of malicious symptoms in clouds via monitoring virtual machines. In Yulei Wu, Geyong Min, Nektarios Georgalas, Jia Hu, Luigi Atzori, Xiaolong Jin, Stephen A. Jarvis, Lei (Chris) Liu, and Ramón Agüero Calvo, editors, *15th IEEE International Conference on Computer and Information Technology, CIT 2015; 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015; 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015; 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015, Liverpool, United Kingdom, October 26-28, 2015*, pages 1703–1710. IEEE, 2015. doi:10.1109/CIT/IUCC/DASC/PICOM.2015.257.

3 Michael A. Bender, Rathish Das, Martin Farach-Colton, Rob Johnson, and William Kuszmaul. Flushing without cascades. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 650–669. SIAM, 2020. doi:10.1137/1.9781611975994.40.

4 Michael A. Bender, Martin Farach-Colton, and William Kuszmaul. Achieving optimal backlog in multi-processor cup games. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1148–1157. ACM, 2019. doi:10.1145/3313276.3316342.

5 Marijke H. L. Bodlaender, Cor A. J. Hurkens, Vincent J. J. Kusters, Frank Staals, Gerhard J. Woeginger, and Hans Zantema. Cinderella versus the wicked stepmother. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, volume 7604 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2012. doi:10.1007/978-3-642-33475-7_5.

6 Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 617–626. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181985.

7 Mee Yee Chan and Francis Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Trans. Computers*, 41(6):755–768, 1992. doi:10.1109/12.144627.

8 Mee Yee Chan and Francis Y. L. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, 1993. doi:10.1007/BF01187034.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

10 Mattia D'Emidio, Gabriele Di Stefano, and Alfredo Navarra. Bamboo garden trimming problem: Priority schedulings. *Algorithms*, 12(4):74, 2019. doi:10.3390/a12040074.

11 Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, editors, *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017, Proceedings*, volume 10139 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2017. doi:10.1007/978-3-319-51963-0_18.

12 Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010. doi:10.1145/1753171.1753195.

13 R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: a real-time scheduling problem. In *[1989] Proceedings of the Twenty-Second Annual Hawaii International Conference*

on System Sciences. Volume II: Software Track, volume 2, pages 693–702 vol.2, January 1989. doi:10.1109/HICSS.1989.48075.

14 Robert Holte, Louis E. Rosier, Igor Tulchinsky, and Donald A. Varvel. Pinwheel scheduling with two distinct numbers. *Theor. Comput. Sci.*, 100(1):105–135, 1992. doi:10.1016/0304-3975(92)90365-M.

15 William Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1558–1577. SIAM, 2020. doi:10.1137/1.9781611975994.96.

16 Shun-Shii Lin and Kwei-Jay Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19(4):411–426, 1997. doi:10.1007/PL00009181.

17 Edward M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985. doi:10.1137/0214021.

18 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

19 Theodore H. Romer and Louis E. Rosier. An algorithm reminiscent of euclidean-gcd computing a function related to pinwheel scheduling. *Algorithmica*, 17(1):1–10, 1997. doi:10.1007/BF02523234.

Finding Water on Poleless Using Melomaniac Myopic Chameleon Robots

Quentin Bramas
University of Strasbourg, ICUBE, France
bramas@unistra.fr

Pascal Lafourcade
LIMOS, University Clermont Auvergne, Aubière, France
pascal.lafourcade@uca.fr

Stéphane Devismes
Université Grenoble Alpes, VERIMAG, France
Stephane.Devismes@univ-grenoble-alpes.fr

Abstract

In 2042, the exoplanet exploration program,[1] launched in 2014 by NASA, finally discovers a new exoplanet so-called *Poleless*, due to the fact that it is not subject to any magnetism. A new generation of autonomous mobile robots, called M2C (for Melomaniac Myopic Chameleon), have been designed to find water on Poleless. To address this problem, we investigate optimal (*w.r.t.*, visibility range and number of used colors) solutions to the infinite grid exploration problem (IGE) by a small team of M2C robots. Our first result shows that minimizing the visibility range and the number of used colors are two orthogonal issues: it is impossible to design a solution to the IGE problem that is optimal *w.r.t.* both parameters simultaneously. Consequently, we address optimality of these two criteria separately by proposing two algorithms; the former being optimal in terms of visibility range, the latter being optimal in terms of number of used colors. It is worth noticing that these two algorithms use a very small number of robots, respectively six and eight.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Luminous Robots, Grid, Infinite Exploration, Treasure Search Problem

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.6

Supplementary Material `https://doi.org/10.5281/zenodo.3606387`

Funding This study was partially supported by the French ANR projects ANR-16-CE40-0023 (DESCARTES) and ANR-16 CE25-0009-03 (ESTATE).

1 Introduction

Poleless is a far-off exoplanet discovered in 2042 that is not subject to any magnetism. Hence, all terrestrial compasses are ineffective on this planet. After the discovery of Poleless, the *National Aeronautics and Space Administration (NASA)* has decided to launch a robotic spacecraft mission toward it in order to evaluate the possibility of a future human presence. For this purpose, they have designed a new generation of autonomous mobile robots called *M2C*, for Melomaniac Myopic Chameleon. These robots are *melomaniac*: in order to synchronously move and to easily coordinate their actions they continuously play and listen the same melody. Of course, the choice of the right song was critical. After a huge campaign of experiments, an international expert panel (notably including several Nobel prizes) has selected the song “Heigh-ho” of the Seven Dwarfs.[2] The M2C robots are also *myopic*, i.e.,

[1] `https://exoplanets.nasa.gov/`
[2] See `https://www.youtube.com/watch?v=HIOxOKYChq4`

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 6; pp. 6:1–6:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

they are equipped with visibility sensors of typically small range. The choice of this feature has been led by several concerns, mainly reducing both the manufacturing costs (still in 2042, NASA has to endure important cuts in its budget) and the energy consumption (of course, no human intervention can be envisioned to recharge their batteries). Additionally, the researchers thought to these two technologies (myopia and melomania) to make the design of the robots as simple as possible. Indeed, simplicity usually implies more robustness and allows to decrease the weight of robots by avoiding the use of fancy, heavy, and costly components. Finally, the M2C robots are *chameleons* meaning that they have the ability to change their color whenever they want. These colors are used for two main reasons:

1. robot intercommunication, since the colors are captured by the visibility sensor of others robots in their surroundings, and
2. persistent memory; actually, colors are the only available persistent memory.

Notice since any modulation in the melody playing may cause an irreversible disynchronization, robots can only use their colors to exchange information.

By analyzing the light spectrum of Poleless, researchers have established the presence of water and a breathable atmosphere with high probability. However, in order to be sure of these facts, an exploration mission is mandatory. Once robots will have landed on the planet, it will be easy for them to test the chemical composition of the atmosphere. Now, to confirm the presence of water, they will have to explore exhaustively the ground of Poleless. Especially since the second important goal of the mission is to find an appropriate place, near a water source, where a future human mission could land. Again, for cost issues, only a typically small team of M2C robots can be used to achieve this task. Basically, they have to coordinate together to explore the planet until (at least) one of them find water. Once it will happen, the robot will both stop moving and singing (informing then the others of the task completion so that they all stop in turn), and send a signal to Earth in order to be precisely localized.

The exact size and the relief of Poleless is unknown, even if it seems to be quite flat. Hence, the surface of Poleless is conveniently discretized as grid of *unbounded* size, where nodes represent locations that can be sensed by robots and edges represent the possibility for a robot to move from one location to another. Hence, the task to be solved by the team of robots is the *treasure search problem* in a grid of unbounded size [11]. Now, this problem is known to be equivalent to the *Infinite Grid Exploration* (IGE) problem [7], which requires each node of an *infinite* grid to be visited within finite time by at least one robot.

We have decided to answer the NASA call for bids by designing new solutions to the IGE problem that are well-suited to the NASA requirements. Notice, in particular, that despite the scientific progress, physical boundaries are still in the agenda in 2042. Namely, the M2C robots are opaque, i.e., a robot is able to see another robot if and only if no other robot lies in the line segment joining them. Moreover, any solution to the IGE problem should achieve *exclusiveness* [2], meaning that, in the grid, any two robots cannot simultaneously occupy the same node nor traverse the same edge. Indeed, even if in 2042 holograms and teleportation techniques are commonly used, they are still not mature enough to be used in a long distance spacecraft mission.

1.1 Contribution

We address the NASA call for bids by investigating low-cost solutions to the IGE problem, i.e., we try as much as possible to limit the necessary visibility range, the number of used colors, and the size of the team.

We first show that minimizing the visibility range and the number of used colors are two orthogonal issues: it is impossible to design an algorithm solving the IGE problem that does not use different colors and that assumes visibility range 1.

Hence, we address the optimality of these two criteria separately. Precisely, we provide two algorithms for the IGE problem:

1. The first algorithm uses only six M2C robots with visibility range 1 and three colors.
2. The second algorithm uses only eight M2C robots with visibility range 2, yet no color (in other word, it assumes *oblivious anonymous* robots).

1.2 Roadmap

In the next section, we define the model associated to M2C robots and Poleless. In Section 3, we present our impossibility result. In Section 4, we describe our two algorithms. Section 5 is dedicated to related work. We conclude in the last section.

2 Preliminaries

As justified in the introduction, we model the ground of Poleless by an *infinite grid* with vertex set in $\mathbb{Z} \times \mathbb{Z}$, i.e., there is an edge between two nodes (i, j) and (k, l) if and only if the *Manhattan distance* between those two nodes, i.e., $|i - k| + |j - l|$, is one. The coordinates are used for the analysis only, i.e., robots cannot access them.

We assume a team $\mathcal{R}$ of $n > 0$ M2C robots evolving on (nodes of) the grid. Recall that M2C robots are melomaniac, i.e., they compute and move *synchronously* by continuously singing Heigh-ho. Precisely, at each beat (or *round*), they all perform an atomic *dance step* as follows. First, they look at their surroundings. Then, they compute a destination among their current position and the four neighboring ones. Finally, they move to the computed destination.

M2C robots are chameleons, i.e., they may change their color, which can be seen by other robots in their surroundings. Let Cl be the set of possible colors. Recall that robots cannot simultaneously occupy the same node nor traverse the same edge. In such a context, a node is *occupied* when a robot is located at the node, otherwise it is *empty*. The *state* of a node is either the color of the robot located at this node, if it is occupied, or $\perp$ otherwise. When a robot looks around, it can only see the states of the node that are within distance $\Phi \in \mathbb{N}^*$ from its position. Φ is called the *visibility range* of the robots. The value of Φ depends on the quality (and so the price) of the robots' lenses. If the visibility range is one, a robot sees its own location and the four neighboring nodes of the grid. After looking around, a robot computes the next destination based only on what it sees and on its own color. During the compute phase, a robot may also decide to change its color.

2.1 Configurations

A *configuration* C is a set of pairs (p, c) where $p \in \mathbb{Z} \times \mathbb{Z}$ is an occupied node and $c \in \mathit{Cl}$ is the color of the robot located at p. A node p is empty if and only if $\forall c$, $(p, c) \notin C$. We sometimes just write the set of occupied nodes when the colors are clear from the context. Also, by a slight abuse of notation, we sometimes partition the configuration into several subsets $C_1, \ldots, C_k$ and write $C = \{C_1, \ldots, C_k\}$ instead of writing $(C = C_1 \cup \ldots \cup C_k) \wedge (\forall i \neq j, C_i \cap C_j = \emptyset)$.

2.2 Views

We denote by G_r the *globally oriented view* centered at Robot r, i.e., the subset of the configuration containing the states of the nodes at distance at most Φ from r, translated so that the coordinates of r is $(0,0)$. We use this globally oriented view in our analysis to describe the movements of the robots: when we say "the robot moves Up", it is according to the globally oriented view. However, since M2C robots are designed to explore Poleless, they do not have any compass and so, they have no access to the globally oriented view. When a robot looks at its surroundings, it obtains a *local view.* To model the absence of compass, we assume that any *local view* acquired by a robot r is the result of an arbitrary *indistinguishable transformation* on G_r. The set $\mathcal{IT}$ of indistinguishable transformations contains:

1. the rotations of angle 0 (to have the identity), $\pi/2$, π and $3\pi/2$, centered at r,
2. the mirroring (robots cannot distinguish between clockwise and counterclockwise), and
3. any combination of rotation and mirroring.

Moreover, since robots may obstruct visibility, the function that removes the state of a node u if there is another robot between u and r is *systematically* applied to obtain the local view. Here, we assume that robots are *self-inconsistent*, meaning that different transformations may be applied at different rounds.

It is important to note that when a robot r computes a destination d, it is relative to its local view $f(G_r)$, which is the globally oriented view transformed by some $f \in \mathcal{IT}$. So, the actual movement of the robot in the *globally oriented view* is $f^{-1}(d)$. For example, if $d = Up$ but the robot sees the grid upside-down (f is the π-rotation), then the robot moves $Down = f^{-1}(Up)$. In a configuration C, $V_C(i,j)$ denotes the globally oriented view of a robot located at (i,j).

2.3 Algorithm

An algorithm $\mathcal{A}$ is a tuple (Cl, I, T) where Cl is the set of possible colors, I is the initial configuration, and T is the transition function $Views \to \{Idle, Up, Left, Down, Right\} \times Cl$, where $Views$ is the set of local views. When the robots are in Configuration C, the configuration C' obtained after one round satisfies: for all $((i,j),c) \in C'$, there exists a robot in C with color $c' \in Cl$ and a transformation $f \in \mathcal{IT}$ such that one of the following conditions holds:

- $((i,j),c') \in C$ and $f^{-1}(T(f(V_C(i,j)))) = (Idle, c)$,
- $((i-1,j),c') \in C$ and $f^{-1}(T(f(V_C(i-1,j)))) = (Right, c)$,
- $((i+1,j),c') \in C$ and $f^{-1}(T(f(V_C(i+1,j)))) = (Left, c)$,
- $((i,j-1),c') \in C$ and $f^{-1}(T(f(V_C(i,j-1)))) = (Up, c)$, or
- $((i,j+1),c') \in C$ and $f^{-1}(T(f(V_C(i,j+1)))) = (Down, c)$.

We denote by $C \mapsto C'$ the fact that C' can be reached in one round from C (*n.b.*, $\mapsto$ is then a binary relation over configurations). An execution of Algorithm $\mathcal{A}$ is then a sequence $(C_i)_{i\in\mathbb{N}}$ of configurations such that $C_0 = I$ and $\forall i \geq 0$, $C_i \mapsto C_{i+1}$.

2.4 Poleless Exploration

An algorithm $\mathcal{A}$ solves the *Poleless exploration* if for every execution $(C_i)_{i\in\mathbb{N}}$ of $\mathcal{A}$ and every node $(i,j) \in \mathbb{Z} \times \mathbb{Z}$ of the grid, there exists $t \in \mathbb{N}$ such that (i,j) is occupied in C_t.

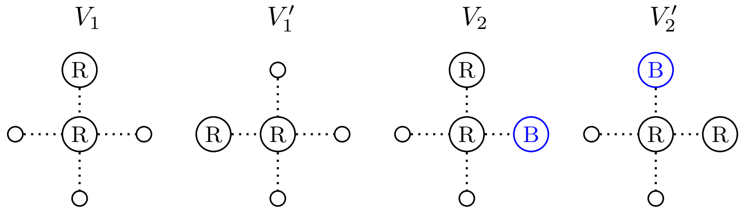


Figure 1 Example of four views. V_1 and V'_1 are indistinguishable. Similarly, V_2 and V'_2 are indistinguishable.

2.5 An Algorithm as a Set of Rules

We write an algorithm as a set of rules, where *a rule* is a triplet $(V, d, c) \in \mathit{Views} \times \{\mathit{Idle}, \mathit{Up}, \mathit{Left}, \mathit{Down}, \mathit{Right}\} \times \mathit{Cl}$.

We say that an algorithm (Cl, I, T) includes the rule (V, d, c), if $T(V) = (d, c)$. By extension, the same rule applies to indistinguishable views, i.e., $\forall f \in \mathcal{IT}, T(f(V)) = (f(d), c)$. Consequently, we forbid an algorithm to contain two rules (V, d, c) and (V', d', c') such that $V' = f(V)$ for some $f \in \mathcal{IT}$.

As an illustrative example, consider local views given in Figure 1. A rule R can associate View V_1 with the direction *Up*. Since *Up* is relative to the view, it means for the robot “I move towards the only robot I see”. View V'_1 is obtained by rotation from V_1, so a robot cannot distinguish V_1 and V'_1, so the same rule R applies in V'_1 and the robot moves Left towards the only robot it sees. However, if in V_1 a robot decides to move to the right towards an empty node, then, since it does not distinguish its right from its left, the actual destination between left and right will be decided according to the applied indistinguishable transformation $f \in \mathcal{IT}$. Similarly, Views V_2 and V'_2 are indistinguishable for the robots (one is the mirror of the other), so any rule that applies to V_2 also applies to V'_2, and conversely. For example, if a robot decides to move towards its blue neighbor B in V_2, it will also move towards its blue neighbor in V'_2.

2.6 Well-defined Algorithms

Recall that robots are assumed to be self-inconsistent. In this context, we say that an algorithm (Cl, I, T) is *well-defined* if the global destination computed by a robot does not depend on the applied indistinguishable transformation f, i.e., for every globally oriented view V, and every transformation $f \in \mathcal{IT}$, we have $T(V) = f^{-1}(T(f(V)))$. Every algorithms we will propose will be well-defined. However, to be as general as possible, we will not make such an assumption in our impossibility results. Finally, remark that a well-defined algorithm has a unique execution.

2.7 Notations

$\vec{t}_{(i,j)}(C)$ denotes the translation of the configuration C of vector (i, j).

3 Impossibility Result

3.1 The Fence Crossing Lemma

In order to explore Poleless, M2C robots regularly cross what we call *fences*. A *fence* L is composed of two infinite adjacent vertical lines $L = (l_1, l_2)$ with $l_1 = \{(i_L, j) | j \in \mathbb{Z}\}$ and $l_2 = \{(i_L + 1, j) | j \in \mathbb{Z}\}$, for some $i_L \in \mathbb{Z}$, such that each robot is initially located at some coordinates (x_0, y_0) satisfying $x_0 < i_L$; see Figure 2. Informally, this means that a fence is made of two infinite adjacent vertical lines that are initially at the right of all robot's positions.

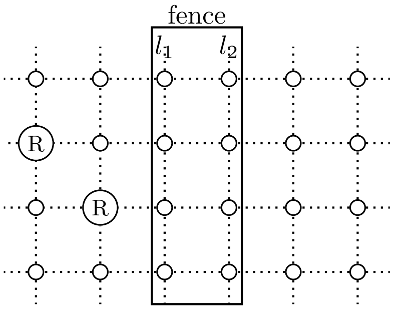
We say that a set of robots have crossed a fence when they are all at the right of the fence at a given time; see Figure 3. Notice that this does not mean that the robots always stays on the right of the fence afterward.

Formally, we say that a set of robots S *has crossed the fence* $L = (l_1, l_2)$ *at Round* t if there exists $t' \leq t$ such that every robot $r \in S$ is located at some coordinates (x_1, y_1) with $x_1 > i_L + 1$ at Round t'.

We say a set of robots S *single-handed crosses the fence* L *between* t *and* t' if for every robot $r \in S$, (1) r is located at some coordinates (x_0, y_0) satisfying $x_0 < i_L$ at Round t (see Figure 2); (2) r is located at some coordinates (x_1, y_1) with $x_1 > i_L + 1$ at Round t' (see Figure 3); and (3) only robots of S are within distance one of r between Round t and Round t'.

We say that a set of robots S *has single-handed crossed* the fence L at Round t if $\exists t' < t'' \leq t$ such that S single-handed crosses the fence $L = (l_1, l_2)$ between t' and t''.

To be more general, we now consider any algorithm, i.e., well-defined or not. We first prove that if robots explore Poleless, then there is a fence that is *single-handed crossed* by a subset of robots; see Lemma 1. This latter result will be used to show that, if robots are anonymous and cannot change their color, the Poleless exploration is impossible under visibility range 1, whatever the number of robots is; see Theorem 2.



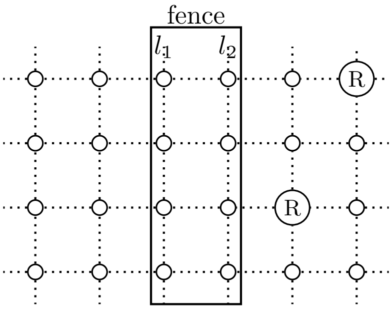


Figure 2 A team of robots in front of a fence. **Figure 3** A team of robots has crossed a fence.

▶ **Lemma 1** (The test of the fence). *If n robots can explore Poleless, then in every execution there exists a fence L and a subset of robots S such that S single-handed crosses L within a finite number of rounds.*

Proof. If n robots successfully explore Poleless, then any node is eventually visited by at least a robot. So, we can choose a node $u = (i, j)$ where i is arbitrarily large: u should be visited within finite number of rounds despite an arbitrary number of fences have to be crossed before. If there is an execution where no subset of robots single-handed crosses at least one of them, then this means that each time a fence is crossed in the execution, some robots are not crossing and are left behind. If i is large enough ($i > n$), then there is not enough robots to cross all the fences to reach u. Hence, a subset of robots single-handed crosses a fence in a finite number of rounds. ◀

3.2 The Impossibility Result

With only one color and under visibility one, at each round there is at most six possible (local) views since every node should contain at most one robot (by exclusiveness), see Figure 4. One can see that any rule associated with view V_0 and a non-idle movement is ambiguous, i.e., the destination depends on the indistinguishable transformation applied to the view. Indeed, the robot in V_0 has no way to distinguish between the four neighboring nodes. The same is true for V_2, V_2', and V_4. Now, as we do not require algorithms to be well-defined, an algorithm may include some ambiguous rules. Actually, there are only two views that can result in non-ambiguous non-idle movement: V_1 where a robot sees only one robot around it and V_3 where a robot sees three robots around it. We denote by R_1^{in}, resp. R_1^{out}, the rule that orders a robot with view V_1 to move towards the neighboring robot, resp. away from the neighboring robot. Similarly, we denote by R_3^{in}, resp. R_3^{out}, the rule that orders a robot with view V_3 to move towards the center robot, resp. towards the empty node; see Figure 5. Note that, since R_1^{in} and R_1^{out} (resp. R_3^{in} and R_3^{out}) are associated with the same view, they cannot be part of the same algorithm.

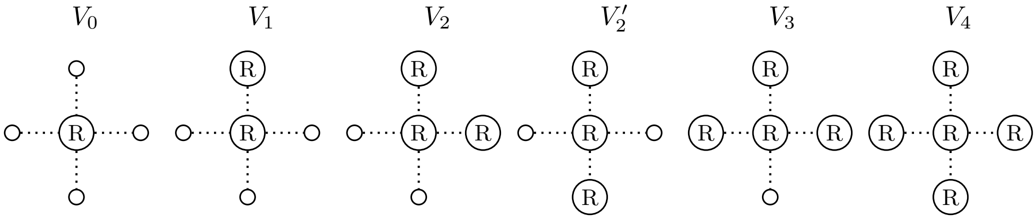


Figure 4 The possible views of a robot with visibility one and without colors.

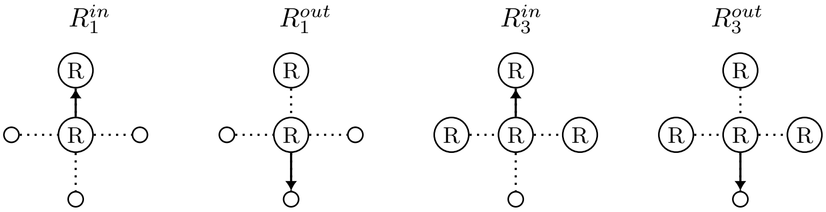


Figure 5 Non-ambiguous and non-idle rules, with visibility one and no color.

▶ **Theorem 2.** *There is no algorithm that solves the Poleless exploration problem with single-color robots and assuming visibility range one.*

Proof. Assume, by contradiction, that an algorithm $\mathcal{A}$ solves the Poleless exploration problem with single-color robots and assuming visibility range one. We show the contradiction by proving that using $\mathcal{A}$, the robots fail the test of the fence (Lemma 1).

To that goal, we first construct an execution by choosing carefully which indistinguishable function is applied to views that are associated with ambiguous rules. If a robot r has a view V where an ambiguous rule applies we do the following:

1. if $V = V_0$, then we apply f such that the global destination is Left.
2. if $V = V_2$, and the rule dictates the robot to move toward an empty node, then we apply f such that the global destination is the unique empty node that is either Up or Down.
3. if $V = V_2$, and the rule dictates the robot to move toward an occupied node, then we apply f such that the global destination is the unique occupied node that is either Up or Down.
4. if $V = V_2'$, and the rule dictates the robot to move toward an empty node, then we apply f such that the global destination is the unique empty node that is either Up or Left.
5. if $V = V_2'$, and the rule dictates the robot to move toward an occupied node, then we apply f such that the global destination is the unique occupied node that is either Up or Left.
6. if $V = V_4$, then we apply f such that the global destination is Left.

We will see that $\mathcal{A}$ cannot contain ambiguous rules for V_1 and V_3. By choosing those indistinguishable transformations, we obtain a unique execution E. According to Lemma 1, there exists a fence $L = (l_1, l_2)$ and a subset of robots S such that S has single-handed crossed L at time t.

By Definition, robots in S are initially located on the left of the fence. We define the Round t_1, resp. t_2, as the last round, before t, when there is a robot of S on l_1, resp. on l_2. Hence, we have, $t_1 < t_2 < t$.

Claim 1: *$\mathcal{A}$ includes at least one* out*-rule, i.e., R_1^{out} or R_3^{out}.*

Proof of the claim: The first robots that enter l_1 move Right (in the global view) towards empty nodes. Moreover, they do so using a non-ambiguous rule since the chosen indistinguishable transformation forces any robot with such rules to move either Up, Down or Left. Thus, $\mathcal{A}$ must include at least one *out*-rule, i.e., R_1^{out} or R_3^{out}.

Claim 2: *$\mathcal{A}$ includes at least one* in*-rule, i.e., R_1^{in} or R_3^{in}.*

Proof of the claim: At Round t_2, all the robots on l_2 move Right to complete the fence-crossing. Again, they do so using a non-ambiguous rule since the chosen indistinguishable transformation forces any robot with such rules to move either Up, Down or Left. Thus, $\mathcal{A}$ must include at least one *in*-rule, i.e., R_1^{in} or R_3^{in}.

Claim 3: *$\mathcal{A}$ includes Rules R_1^{in} and R_3^{out}, but neither R_3^{in} nor R_1^{out}.*

Proof of the claim: Since an algorithm cannot have two rules based on the same view, $\mathcal{A}$ either includes Rules R_1^{in} and R_3^{out}, or Rules R_3^{in} and R_1^{out}, by Claims 1 and 2. So, assume, by contradiction, that $\mathcal{A}$ includes R_3^{in} and R_1^{out}, but neither R_1^{in}, nor R_3^{out}. At Round t_2, all robots on l_2 (at least one) leave it. Again, in this case, these robots necessarily execute a non-ambiguous rule: the only available rule is R_3^{in}. Yet, this implies that there is an infinite chain of robots on l_2, which contradicts the fact that there is a finite number of robots.

Using Claim 3 we can show the following Claim.

Claim 4: *There are two adjacent robots r_a and r_a' (of S) on l_2 at Round $t_1 + 1$.*

Proof of the claim:

At Round t_1, let r_a be any robot on l_1. Then, r_a leaves l_1 towards l_2. Again, r_a should execute a non-ambiguous rule at Round t_1, i.e., R_1^{in}, by Claim 3. So, r_a moves towards a robot r_d. This implies that $r_d \in S$ is not idle at Round t_1 since otherwise this would create a collision, violating then exclusiveness. So, r_d has only the three following possibilities at Round t_1: (a) r_d executes R_1^{in} or an ambiguous rule toward an occupied node, (b) r_d executes an ambiguous rule towards an empty node or (c) r_d executes Rule R_3^{out}. We now show that in all theses cases, we either obtain a contradiction, or we show that there are two adjacent robots r_a and r'_a (of S) on l_2 at Round $t_1 + 1$.

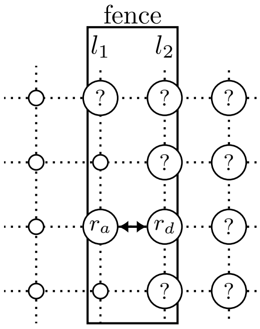


Figure 6 Case (a), reaching a contradiction.

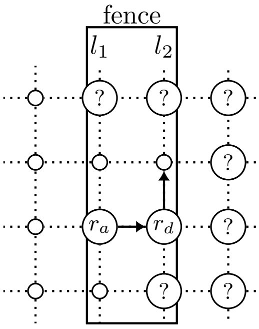


Figure 7 Case (b), r_a and r_d are neighbors at Round $t_1 + 1$.

- In Case (a), illustrated in Figure 6, if R_1^{in} is executed by r_d, then r_a and r_d exchange their positions, violating then exclusiveness, a contradiction. If an ambiguous rule orders r_d to move towards an occupied destination, then there is an indistinguishable transformation that makes move r_d to the Left. Hence, there is a possible execution that behaves as E until Round $t_1 - 1$, but where r_a and r_d exchange their positions during Round t_1, violating then exclusiveness, a contradiction.
- In Case (b), illustrated in Figure 7, r_d sees either V_2 or V'_2 (the only ambiguous views with at least one occupied neighbor and one empty neighbor). So r_d has two neighbors, one of which is r_a. So, the chosen indistinguishable transformation makes it moves Up or Down towards an empty node on l_2 and becomes a neighbor of r_a at Round $t_1 + 1$. So, by letting $r_d = r'_a$, we obtain that there are two adjacent robots r_a and r'_a (of S) on l_2 at Round $t_1 + 1$.

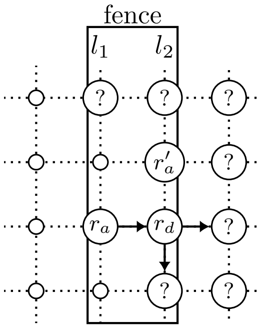


Figure 8 Case (c), r_a and r'_a are neighbors at Round $t_1 + 1$.

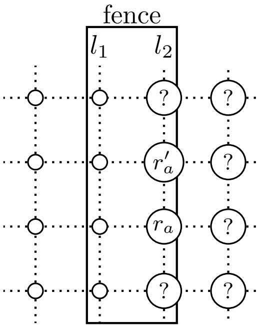


Figure 9 Robots r_a and r'_a are stuck on the fence.

- In Case (c), illustrated in Figure 8, one of r_d's neighbor, denoted r'_a, is also located on l_2. r'_a cannot execute R_1^{in} to move towards r_d, otherwise it would create a collision with r_a, violating then exclusiveness. Also, r'_a does not have a neighbor on l_1 because that would prevent r_a from applying Rule R_1^{in}. So, if R_3^{out} applies to r'_a, it moves towards l_1, contradicting the definition of t_1. An ambiguous rule cannot apply to r'_a either. Indeed, if an ambiguous rule with an empty destination applies, the chosen indistinguishable transformation makes r'_a moves towards l_1 (and so violating the definition of t_1), and if an ambiguous rule with an occupied destination applies, then there is an indistinguishable transformation that makes r'_a move toward r_d. So, again, there is an execution that behaves as E until Round $t_1 - 1$ but where both r_a and r'_a move to the same position during Round t_1, creating a collision with r_a at Round $t_1 + 1$, a contradiction. Hence, r'_a stays idle and r'_a and r_a are adjacent on l_2 at Round $t_1 + 1$, and we are done.

From Claim 4, we have an execution where r_a and r'_a are adjacent on l_2 at Round $t_1 + 1$ (Figure 9). To conclude the proof, we show that if two robots are adjacent on l_2 at Round t' with $t_1 < t' \leq t_2$, then they are adjacent on l_2 Round $t' + 1$. This contradicts the fact all the robots leave l_2 at Round t_2.

When r_a and r'_a are adjacent on l_2 at time t' (with r_a below r'_a), one can observe that R_3^{out} cannot apply to any of them, nor any ambiguous rule with an empty destination, otherwise the chosen transformation would make them move toward l_1, violating the definition of t_1.

Now either r_a executes (i) R_1^{in}, (ii) an ambiguous rule towards an occupied destination, or (iii) stays idle. If r_a executes R_1^{in} or an ambiguous rule towards an occupied destination, it moves UP towards r'_a. r'_a cannot stay idle (since otherwise it would create a collision), and cannot execute R_1^{in}, otherwise it would violates the exclusiveness, so it executes an ambiguous rule toward an occupied destination and moves UP. At Round $t' + 1$, r_a and r'_a are still adjacent on l_2. If r_a stays idle, r'_a cannot execute R_1^{in}, otherwise it would create a collision, nor an ambiguous rule towards an occupied destination, otherwise we can construct a possible execution that behaves as E until Round t', but where r'_a moves towards r_a to create a collision, using the appropriate indistinguishable transformation. So r'_a stays idle as well. At Round $t' + 1$, r_a and r'_a are still adjacent on l_2.

This contradicts the fact that all the robots on l_2 at Round t_2 move Right. In the execution E, fence L is never single-handed crossed, which contradicts our initial assumption. ◀

4 Algorithms

In this section, we give two algorithms, respectively called $\mathcal{A}^{1}_{(6,3)}$ and $\mathcal{A}^{2}_{(8,1)}$, for solving the Poleless exploration. Algorithm $\mathcal{A}^{1}_{(6,3)}$ (presented in Subsection 4.1) assumes visibility one and uses six robots and three colors. Algorithm $\mathcal{A}^{2}_{(8,1)}$ requires visibility two and uses eight anonymous oblivious robots, i.e., indistinguishable robots. The animations of these two algorithms are available in our complementary material [6]. The fact that the rules of these algorithms are well-defined has been checked by the script that generated those animations. This has been done by making sure that (1) the view of any rule cannot be transformed into the view of another rule using mirroring, rotation, or a combination of the two, and (2) for each rule, the global destination does not depend on the applied local indistinguishable transformation.

4.1 Six Robots with Three Colors under Visibility Range One

The six robots are divided into two categories: the *beacon* robots and the *moving group*. There are four beacon robots, each of those being B-colored in the following. The *moving group* is made of two robots: one L-colored *leader* and one F-colored *follower*. However, some robots change their role (by changing their color) along the execution.

Initially, all the robots are close together and organized as shown in Figure 10. The beacons are used to delimit the area which has been already explored. The moving group aims at reaching the beacons one by one. Each time the moving group reaches a beacon,

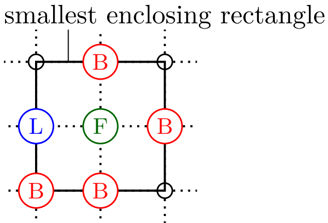


Figure 10 Initial configuration of Algorithm $\mathcal{A}^{1}_{(6,3)}$.

robots make an *adjustment*. At the end of the adjustment, the new beacon position is in the diagonal (two hops) of the previous one and the moving group has made a turn toward the next beacon. This adjustment, in particular, allows to take the newly explored nodes into account. The moving group then continues toward the next beacon, and so on. Each time the moving group comes back to the first beacon, a so-called *phase* terminates: the border of the area initially delimited by the four beacons is now fully visited, and the area newly delimited by the beacons is bigger; see Figure 11 to visualize the increasing area that is explored by the moving group.

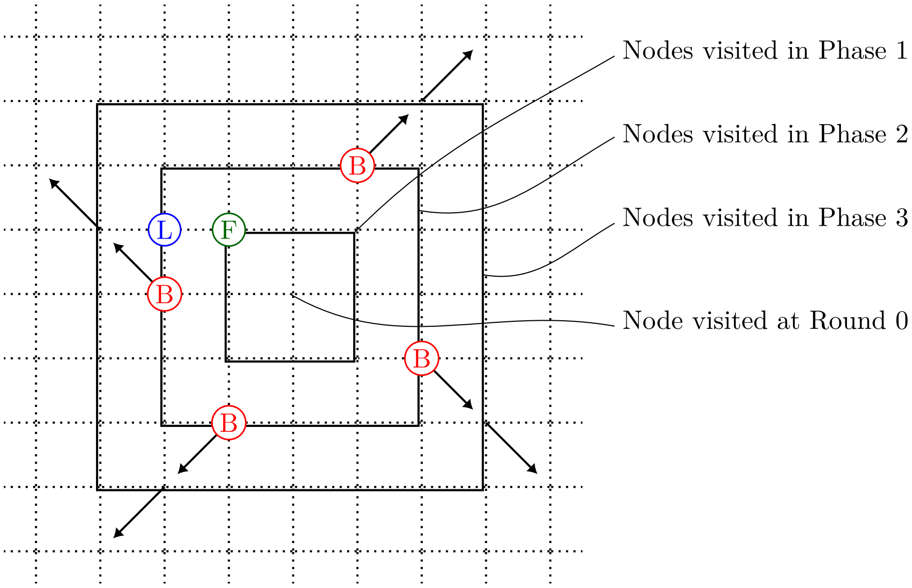


Figure 11 Visited area after the first three phases for $\mathcal{A}^{1}_{(6,3)}$. The positions of the robots at those at the beginning of the second phase.

During an *adjustment*, the leader becomes the beacon, the beacon becomes the leader, and after that, the new moving group travels toward the next beacon. Figure 15 shows the sequence of moves of an adjustments occurring at the top-right beacon.

The moving group successfully performs a phase independently of the distance between the beacons, so that infinitely many growing phases are achieved in sequence. The Poleless exploration problem is then solved as any node of the grid is eventually included in the area delimited by the beacons. Note that we use the same technique for the second algorithm.

The rules that allow the moving group to travel along a straight line are shown in Figure 12. The rules to make an adjustment are shown in Figure 13. In order for the two first rounds to work as expected, three more rules (Figure 14) are necessary. Those rules are similar to the previous rules, but consider cases where more robots appear in view. This

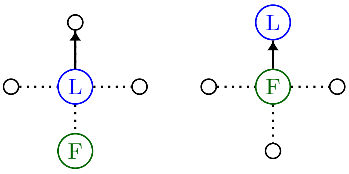


Figure 12 The two rules that make the moving group travel along a straight line.

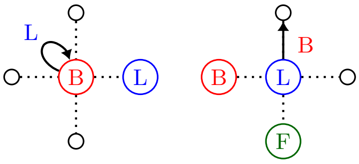


Figure 13 The two rules that perform an adjustment.

actually occurs at the beginning of the algorithm only, when all the robots are close together. For instance, the follower robot should move towards the leader, even if it sees beacon robots around it.

▶ **Theorem 3.** *Algorithm $\mathcal{A}^1_{(6,3)}$ solves the Poleless exploration problem using six robots, three colors and visibility range of one.*

Proof. In the following we assume, w.l.o.g., that Node $(0,0)$ is the one where the bottom-left-most beacon robot is located in the initial configuration; see Figure 10. Recall that these global coordinates are used for the analysis only: robots cannot access those coordinates.

Using this coordinate system, the initial configuration is denoted C^0 and is decomposed as follow: $C^0 = \{M^0, C^0_0, C^0_1, C^0_2, C^0_3\}$, where $M^0 = \{((0,1), L), ((1,1), F)\}$, $C^0_0 = \{((0,0), B)\}$, $C^0_1 = \{((1,0), B)\}$, $C^0_2 = \{((2,1), B)\}$, and $C^0_3 = \{((1,2), B)\}$. We define the configuration $C^i = \{M^i, C^i_0, C^i_1, C^i_2, C^i_3\}$ in Phase i, where $M^i = \vec{t}_{(-i,i)}(M^0)$, $C^i_0 = \vec{t}_{(-i,i)}(C^0_0)$, $C^i_1 = \vec{t}_{(-i,-i)}(C^0_1)$, $C^i_2 = \vec{t}_{(i,-i)}(C^0_2)$, and $C^i_3 = \vec{t}_{(i,i)}(C^0_3)$. Informally, the configuration in Phase i is obtained by diagonally translating i times the positions of the beacons and the moving group in the initial configuration. We now prove that starting from Configuration C^i, Configuration C^{i+1} is eventually reached. Since the initial configuration of our algorithm is C^0, this implies that every configuration C^i, for every $i \geq 0$, is gradually reached. By doing so, the leader robot visits all the edges of growing rectangles. The illustration of one cycle is presented in Figure 16.

Assume we reach the first configuration C^i of Phase i at time t. Recall that $C^i = \{((-i, i+1), L), ((1-i, i+1), F), C^i_0, C^i_1, C^i_2, C^i_3\}$. After one round, the configuration is $\{((-i, i), L), ((-i, i+1), F), C^{i+1}_0, C^i_1, C^i_2, C^i_3\}$.

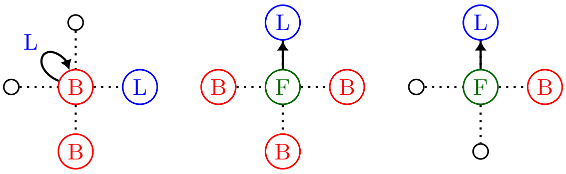


Figure 14 The three rules similar the the previous rules, but used at the beginning of the algorithm, when the robots close together.

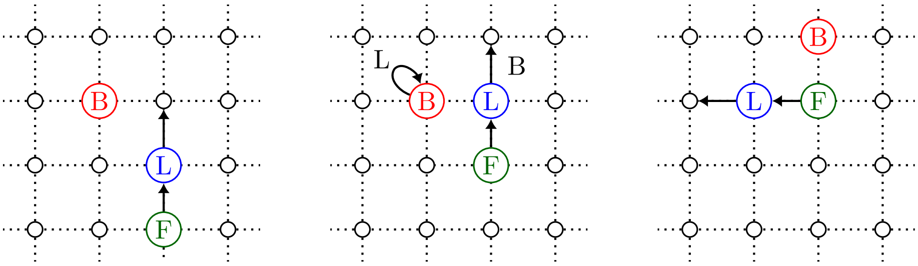


Figure 15 Sequence of moves for an adjustment at the top left beacon robot.

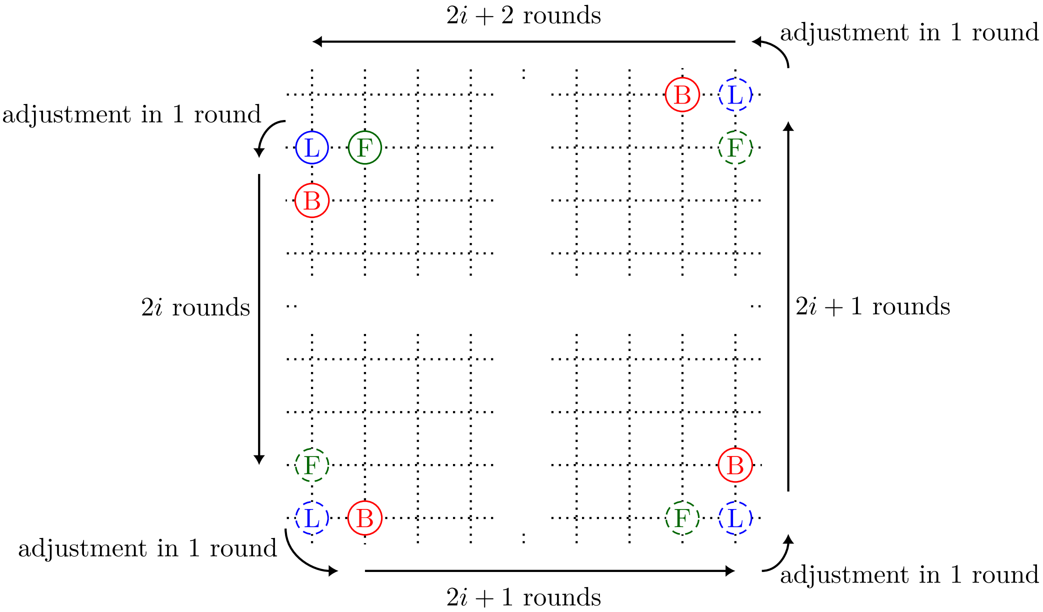


Figure 16 Visualization of one phase.

Then, the moving group travels along a straight line during $2i$ rounds until robot with Color L sees the second beacon robot. Indeed, at time $t+1$, it is located at $(-i,i)$ and the second beacon robot is at $(1-i,-i)$.

At time $t+2i+1$, when the robot with Color L sees the second beacon robot, the second adjustment occurs. At time $t+2i+2$, the configuration is $\{((1-i,-i),L),((-i,-i),F),C_0^{i+1}, C_1^{i+1},C_2^i,C_3^i\}$. Then, the moving group travels during $2i+1$ rounds until it reaches the third beacon robot. Indeed, the robot with Color L is at $(1-i,-i)$ at time $t+2i+2$ and the third beacon is at $(2+i,1-i)$.

When the robot with color L sees the third beacon, the third adjustment occurs and the reached configuration is $\{((2+i,1-i),L),((2+i,-i),F),C_0^{i+1}, C_1^{i+1},C_2^{i+1},C_3^i\}$.

Then, the moving group travels during $2i+1$ rounds until the robot with Color L sees the fourth beacon robot. The last adjustment is performed to obtain the configuration $\{((1+i,2+i),L),((2+i,2+i),F),C_0^{i+1}, C_1^{i+1},C_2^{i+1},C_3^{i+1}\}$. Finally, after $2i+2$ rounds, the moving group comes back to the first beacon robot and the configuration is exactly C^{i+1} at time $t+8i+5$.

Inductively, the robots start from configuration C^0 and reach configuration C^i within finite time, for any $i \geq 0$. Also, Node $(1,1)$ is visited at Round 0, and the set V_i of nodes visited by the robot with Color L between Phase i and $i+1$ contains the edges of the rectangle $\{\vec{t}_{(-i,-i)}(0,0), \vec{t}_{(i,-i)}(2,0), \vec{t}_{(i,i)}(2,2), \vec{t}_{(-i,i)}(0,2)\}$; see Figure 11. Since $\{(1,1)\} \cup \bigcup_{i\geq 0} V_i = \mathbb{Z} \times \mathbb{Z}$, Algorithm $\mathcal{A}^{1}_{(6,3)}$ solves the Poleless exploration problem. ◀

4.2 Eight Anonymous Oblivious Robots under Visibility Two

Algorithm $\mathcal{A}^{2}_{(8,1)}$ is based on principles similar to those used in Algorithm $\mathcal{A}^{1}_{(6,3)}$: four *beacon* robots delimit the visited area, and there is a *moving group*, this time made of four robots, to travel from one beacon to another. The initial configuration of $\mathcal{A}^{2}_{(8,1)}$ is described in Figure 17. Since robots are anonymous, the only way to distinguish them is to use their relative locations. Notice that, this time, beacon robots are always the same. To maintain this property, we ensures that beacon robots are never adjacent to any other robot. Since the visibility range is two, a beacon can see other robots and move before becoming their neighbor.

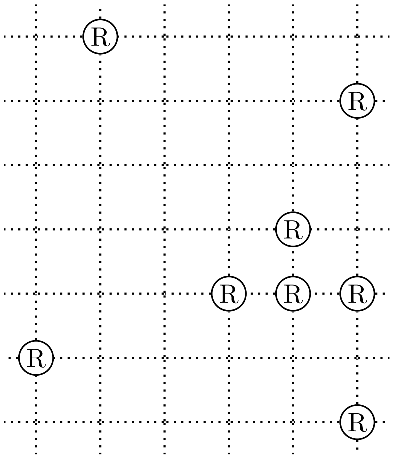


Figure 17 Initial configuration I of $\mathcal{A}^{2}_{(8,1)}$.

Observe that since the visibility range is two, the obstructed visibility can impact the local view of a robot because a robot at distance one can hide a robot behind it at distance two. So, the rules of $\mathcal{A}^{2}_{(8,1)}$ should not depend on the states of the nodes that are hidden by a robot. To make it clear, those nodes will be crossed out in the illustrations of our rules; see, e.g., Figure 18.

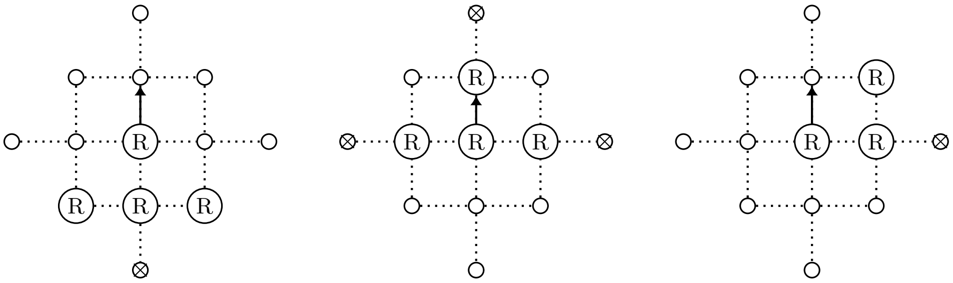


Figure 18 Rules to move along a straight line.

The first three rules (see Figure 18) allow the moving group to move along a straight line. The moving group always forms a *spaceship shape* where one robot is at the bow, one robot is at the stern, and there is one robot on each side, adjacent to the stern. When in formation, each robot knows whether it is at the bow, the stern, or at a side of the spaceship. However, the robots on the side do not know on which side they are, since there is no common chirality. The first rule orders the bow robot to move away from the other robots, the second rule orders the stern robot to move towards the bow robot, and the third rule orders the side robots to move to the same direction as the stern robot.

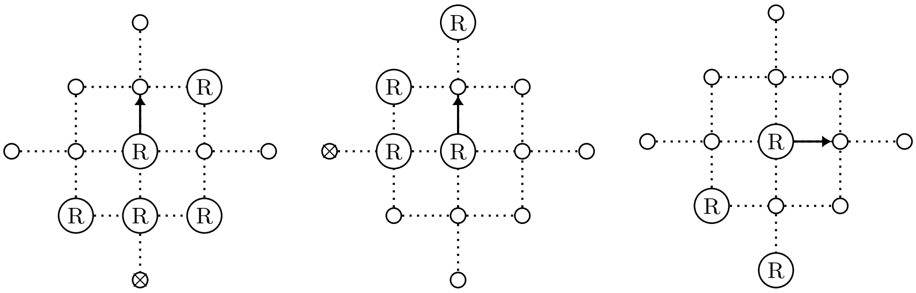


Figure 19 Rules to make the spaceship moving along a straight line when seeing the beacon, and to make the beacon move away.

Then, when the moving group meets a beacon robot, an adjustment is made in two rounds. The first round of the adjustment, robots execute the rules defined in Figure 19. The first two rules order the moving group to act as if the beacon was not there i.e., they continue to move in the same direction. The third rule orders the beacon to move away from the bow robot. The beacon robot can distinguish the correct direction because it also sees a side robot. In the second round of the adjustment, the two rules given in Figure 20 are used. The first rule orders the bow robot to continue as usual (i.e., as if the beacon was not here) and the second rule orders the side robot that sees the beacon robot in diagonal to move towards the stern robot. After the execution of those rules, the spaceship shape is preserved, but the bow robot has become a side robot, and this side robot has become the bow robot. In the same round, the beacon robot executes the same rule as in the first round of the adjustment (the third rule of Figure 19) to move away from the group. The view is mirrored from the first round of the adjustment, so after the two rounds of the adjustment, the beacon has moved diagonally.

The last rule given in Figure 21 is necessary to make moving as expected the side robot that still sees the beacon robot right after an adjustment.

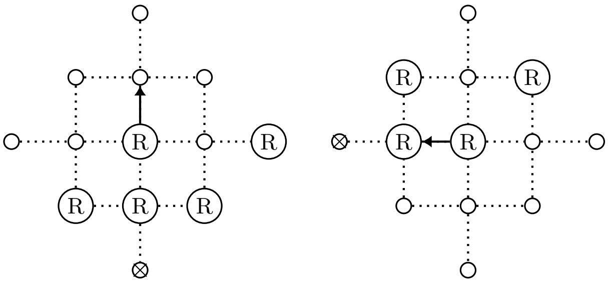


Figure 20 The leader moves in straight line again to become a side follower, the side follower that sees the beacon moves left (the beacon move away again using the same rule as before).

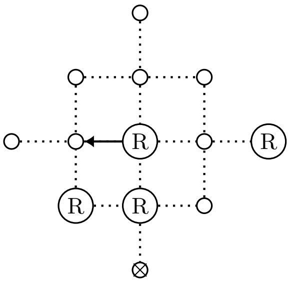


Figure 21 The moving group moves away from the beacon. The side follower that still see the beacon moves away from it.

▶ **Theorem 4.** *Algorithm $\mathcal{A}^2_{(8,1)}$ solves the exclusive Poleless exploration problem using eight robots without color and visibility range of two.*

Proof. The proof of this theorem is similar to the proof of Theorem 4: we decompose the execution into phases, show by induction that each phase is eventually reached, and finally a particular rectangle is visited during each phase.

We fix a global coordinate system, not accessible to the robots, where the initial configuration, denoted by C^0, is split as follows: $C^0 = \{M^0, C^0_0, C^0_1, C^0_2, C^0_3\}$, where $M^0 = \{(3,2),(4,2),(5,2),(4,3)\}$, $C^0_0 = \{(5,5)\}$, $C^0_1 = \{(1,6)\}$, $C^0_2 = \{(0,1)\}$, and $C^0_3 = \{(5,0)\}$. We define the configuration $C^i = \{M^i, C^i_0, C^i_1, C^i_2, C^i_3\}$ in Phase i, where $M^i = \vec{t}_{(i,i)}(M^0)$, $C^i_0 = \vec{t}_{(i,i)}(C^0_0)$, $C^i_1 = \vec{t}_{(-i,i)}(C^0_1)$, $C^i_2 = \vec{t}_{(-i,-i)}(C^0_2)$, and $C^i_3 = \vec{t}_{(i,-i)}(C^0_3)$.

Here, C^i_0 contains the first beacon robot visited in Phase i, located at the upper right corner of the configuration.

Assume we reach the first configuration C^i of Phase i at time t. Recall that $C^i = \{(i+3,i+2),(i+4,i+2),(i+5,i+2),(i+4,i+3), C^i_0, C^i_1, C^i_2, C^i_3\}$. After three rounds, the configuration is $\{(i+3,i+5),(i+4,i+4),(i+4,i+5),(i+4,i+6), C^{i+1}_0, C^i_1, C^i_2, C^i_3\}$.

Then, the moving group has to travel along a straight line during $2i+1$ rounds until the bow robot sees the second beacon robot. Indeed, at time $t+3$, the bow robot is located at $(3+i, 5+i)$ and the second beacon robot is at $(1-i, 6+i)$.

At time $t+2i+4$, when the bow robot sees the second beacon robot, the second adjustment occurs. At time $t+2i+6$, the configuration is $\{(1-i,i+4),(-i,i+5),(1-i,i+5),(2-i,i+5), C^{i+1}_0, C^{i+1}_1, C^i_2, C^i_3\}$. Then, the moving group travels during $2i+2$ rounds until it reaches the third beacon robot. Indeed, the bow robot is at $(1-i, 4+i)$ at time $t+2i+4$ and the third beacon is at $(-i, 1-i)$.

This continues until the configuration C^{i+1} is reached.

Inductively, the robots start from configuration C^0 and reach configuration C^i within finite time, for any $i \geq 0$. The set V_i of nodes visited between Phase i and $i+1$ includes the edges of the rectangle $\{\vec{t}_{(-i,-i)}(0,1), \vec{t}_{(i,-i)}(5,1), \vec{t}_{(i,i)}(5,5), \vec{t}_{(-i,i)}(0,5)\}$. Also, the set V_0 contains the nodes inside rectangle $\{(0,1),(5,1),(5,5),(0,5)\}$ as they are visited during the first phase. Since $\bigcup_{i\geq 0} V_i = \mathbb{Z}\times\mathbb{Z}$, our algorithm solves the Poleless exploration problem. ◀

5 Related Work

The robots we have considered are known as *luminous robots* in the literature. They have been introduced by Peleg in [14]. In [8], the authors compare the computational power of luminous robots with respect to the three main execution models: fully-synchronous, semi-synchronous, and asynchronous. Solutions for dedicated problems such as *weak gathering* or *mutual visibility* have been respectively investigated in [12] and [13].

Exploration tasks have been first considered in the context of finite graphs. In this setting, two main variants, respectively called the *terminating* and *perpetual* exploration, have been considered. The terminating exploration requires every possible location to be eventually visited by at least one robot, with the additional constraint that all robots stop moving after task completion. In contrast, the perpetual exploration requires each location to be visited infinitely often by all or a part of robots. In [9], authors solve terminating exploration of any finite grid using few asynchronous anonymous oblivious robots, yet assuming unbounded visibility range. The exclusive perpetual exploration of a finite grid is considered in the same model in [3].

Various terminating problems have been investigated in infinite grids such as *arbitrary pattern formation* [4], *mutual visibility* [1], and *gathering* [15, 10].

Emek et al. [11] have investigated the treasure search problem in an unbounded size grid [7]. They consider robots operating in two models: the semi-synchronous and synchronous ones. However, they do not impose the exclusivity at all since their robots can only sense the states of the robots located at the same node (in that sense, the visibility range is zero). Moreover, in contrast with our work, they assume all robots agree on a *global compass*, i.e., they all agree on the same directions North-South and East-West. They propose two algorithms that respectively need three synchronous and four semi-synchronous robots. Moreover, they exclude solutions for two robots.

In a followup paper [7], Brandt et al. extend the impossibility result of Emek et al. by showing the impossibility of exploring an infinite grid with three semi-synchronous deterministic robots that agree on a *global compass.*

In [5], we have investigated the IGE problem by a swarm of autonomous mobile luminous synchronous robots. Those robots agree on a common chirality, but have no global compass, while here we neither assumed a common chirality, nor a global compass. Precisely, we show that using only three non-modifiable colors, six robots, with a visibility range one, are necessary and sufficient to solve the IGE problem. We also show that using modifiable colors with only five states, five such robots, with a visibility range one, are necessary and sufficient to solve the IGE problem. Finally, assuming visibility range two, we provide an algorithm that solves the IGE problem using only seven identical robots without light.

6 Conclusion

Thanks to our impossibility results, NASA has been convinced that our two algorithms with Melomaniac Myopic Chameleon Robots are relevant to explore Poleless. Due to some restrictions on the budget, they finally decided to use robots with only a visibility range of one. Then, in 2048, the mission “Finding Water on Poleless” was a great success, and fortunately thanks to our algorithm water was found on Poleless. Once this great news has been known, NASA realizes that the time to send some humans to Poleless the water might have change the place. Hence, they ask a different challenge to the community with the perpetual exploration of Poleless, in order to regularly update information about water localizations With this new challenge our roadmap is clear for the next months.

References

1 Ranendu Adhikary, Kaustav Bose, Manash Kumar Kundu, and Buddhadeb Sau. Mutual visibility by asynchronous robots on infinite grid. In *Algorithms for Sensor Systems - 14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 83–101, 2018.

2 Roberto Baldoni, François Bonnet, Alessia Milani, and Michel Raynal. Anonymous graph exploration without collision by mobile robots. *Inf. Process. Lett.*, 109(2):98–103, 2008. doi:10.1016/j.ipl.2008.08.011.

3 François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In Antonio Fernández Anta, editor, *Proceedings of International Conference on Principles of Distributed Systems (OPODIS 2011)*, number 7109 in Lecture Notes in Computer Science (LNCS), pages 251–265, Toulouse, France, December 2011. Springer Berlin / Heidelberg. URL: http://www.springerlink.com/content/9l3v424157681707/.

4 Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, and Buddhadeb Sau. Arbitrary pattern formation on infinite grid by asynchronous oblivious robots. In *WALCOM: Algorithms and Computation - 13th International Conference, WALCOM 2019, Guwahati, India, February 27 - March 2, 2019, Proceedings*, pages 354–366, 2019.

5 Quentin Bramas, Stéphane Devismes, and Pascal Lafourcade. Infinite grid exploration by disoriented robots. In Michele Flammini Keren Censor-Hillel, editor, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 340–344. Springer, 2019. doi:10.1007/978-3-030-24922-9_25.

6 Quentin Bramas, Stéphane Devismes, and Pascal Lafourcade. Poleless Exploration with Melomaniac Myopic Chameleon Robots: The Animations, January 2020. doi:10.5281/zenodo.3606387.

7 Sebastian Brandt, Jara Uitto, and Roger Wattenhofer. A tight lower bound for semi-synchronous collaborative grid exploration. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.13.

8 Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609(P1):171–184, January 2016. doi:10.1016/j.tcs.2015.09.018.

9 Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Terminating Exploration Of A Grid By An Optimal Number Of Asynchronous Oblivious Robots. *The Computer Journal*, March 2020. doi:10.1093/comjnl/bxz166.

10 Durjoy Dutta, Tandrima Dey, and Sruti Gan Chaudhuri. Gathering multiple robots in a ring and an infinite grid. In *Distributed Computing and Internet Technology - 13th International Conference, ICDCIT 2017, Bhubaneswar, India, January 13-16, 2017, Proceedings*, pages 15–26, 2017.

11 Yuval Emek, Tobias Langner, David Stolz, Jara Uitto, and Roger Wattenhofer. How many ants does it take to find the food? *Theor. Comput. Sci.*, 608(P3):255–267, December 2015. doi:10.1016/j.tcs.2015.05.054.

12 Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. Mutual visibility by luminous robots without collisions. *Inf. Comput.*, 254:392–418, 2017. doi:10.1016/j.ic.2016.09.005.

13 Fukuhito Ooshita and Ajoy K. Datta. Brief announcement: Feasibility of weak gathering in connected-over-time dynamic rings. In *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium, SSS 2018, Tokyo, Japan, November 4-7, 2018, Proceedings*, pages 393–397, 2018.

14 David Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. In *Proceedings of the 7th International Conference on Distributed Computing*, IWDC'05, pages 1–12, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11603771_1.

15 Gabriele Di Stefano and Alfredo Navarra. Gathering of oblivious robots on infinite grids with minimum traveled distance. *Inf. Comput.*, 254:377–391, 2017. doi:10.1016/j.ic.2016.09.004.

1×1 Rush Hour with Fixed Blocks Is PSPACE-Complete

Josh Brunner
Massachusetts Institute of Technology,
Cambridge, MA, USA
brunnerj@mit.edu

Lily Chung
Massachusetts Institute of Technology,
Cambridge, MA, USA
ikdc@mit.edu

Erik D. Demaine
Massachusetts Institute of Technology,
Cambridge, MA, USA
edemaine@mit.edu

Dylan Hendrickson
Massachusetts Institute of Technology,
Cambridge, MA, USA
dylanhen@mit.edu

Adam Hesterberg
Massachusetts Institute of Technology,
Cambridge, MA, USA
achester@mit.edu

Adam Suhl
Algorand, Boston, MA, USA

Avi Zeff
Massachusetts Institute of Technology,
Cambridge, MA, USA
avizeff@mit.edu

Abstract

Consider $n^2 - 1$ unit-square blocks in an $n \times n$ square board, where each block is labeled as movable horizontally (only), movable vertically (only), or immovable – a variation of Rush Hour with only 1×1 cars and fixed blocks. We prove that it is PSPACE-complete to decide whether a given block can reach the left edge of the board, by reduction from Nondeterministic Constraint Logic via 2-color oriented Subway Shuffle. By contrast, polynomial-time algorithms are known for deciding whether a given block can be moved by one space, or when each block either is immovable or can move both horizontally and vertically. Our result answers a 15-year-old open problem by Tromp and Cilibrasi, and strengthens previous PSPACE-completeness results for Rush Hour with vertical 1×2 and horizontal 2×1 movable blocks and 4-color Subway Shuffle.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases puzzles, sliding blocks, PSPACE-hardness

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.7

Related Version This paper is also available on arXiv at `https://arXiv.org/abs/2003.09914`.

Acknowledgements We thank the many colleagues over the years for their early collaborations in trying to resolve the 1×1 Rush Hour problem (when E. Demaine mentioned it to various groups over the years): Timothy Abbott, Kunal Agrawal, Reid Barton, Punyashloka Biswal, Cy Chen, Martin Demaine, Jeremy Fineman, Seth Gilbert, David Glasser, Flena Guisoresac, MohammadTaghi Hajiaghayi, Nick Harvey, Takehiro Ito, Tali Kaufman, Charles Leiserson, Petar Maymounkov, Joseph Mitchell, Edya Ladan Mozes, Krzysztof Onak, Mihai Pătraşcu, Guy Rothblum, Diane Souvaine, Grant Wang, Oren Weimann, Zhong You (MIT, November 2005); Jeffrey Bosboom, Sarah Eisenstat, Jayson Lynch, and Mikhail Rudoy (MIT 6.890, Fall 2014); and Joshua Ani, Erick Friis, Jonathan Gabor, Josh Gruenstein, Linus Hamilton, Lior Hirschfeld, Jayson Lynch, John Strang, Julian Wellman (MIT 6.892, Spring 2019, together with the present authors).

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 7; pp. 7:1–7:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In a ***sliding block puzzle***, the player moves blocks (typically rectangles) within a box (often a rectangle) to achieve a desired configuration. Such puzzles date back to the 15 Puzzle, invented by Noyes Chapman in 1874 and popularized by Sam Loyd in 1891 [13], where the blocks are unit squares. One of the first puzzles to use rectangular pieces is the Pennant Puzzle by L. W. Hardy in 1909, popularized under the name Dad's Puzzle from 1926, whose 10 pieces require a whopping 59 moves to solve [5]. In general, such puzzles are PSPACE-complete to solve, even for 1×2 blocks in a square box [7, 8], which was the original application for the hardness framework Nondeterministic Constraint Logic (NCL). For unit-square pieces (as in the 15 Puzzle), such puzzles can be solved in polynomial time, though finding a shortest solution is NP-complete [10, 3].

In the 1970s, two famous puzzle designers – Don Rubin in the USA and Nobuyuki "Nob" Yoshigahara (1936–2004) in Japan – independently invented [9] a new type of sliding block puzzle, where each block can move only horizontally or only vertically. The motivation is to imagine each block as a car that can drive forward and reverse, but cannot turn; the goal is to get one car (yours) to "escape" by reaching a particular edge of the board. The original forms – Rubin's "Parking Lot" [11] and Nob's "Tokyo Parking" [15] – imagined a poor parking-lot attendant trying to extract a car. Binary Arts (now ThinkFun) commercialized Nob's 6×6 puzzles as ***Rush Hour*** in 1996, where a driver named Joe is "figuring things out on their way to the American Dream" [16]. The physical game design led to a design patent [18] and many variations by ThinkFun since [16].[1] Computer implementations of the game at one point led to a lawsuit against Apple and an app developer [9].

The complexity of Rush Hour was first analyzed by Flake and Baum in 2002 [4]. They proved that the game is PSPACE-complete with the original piece types – 1×2 and 1×3 cars, which can move only in their long direction – when the goal is to move one car to the edge of a square board. In 2005, Tromp and Cilibrasi [17] strengthened this result to use just 1×2 cars (which again can move only in their long direction), using NCL. Hearn and Demaine [8, 6] simplified this proof, and proved analogous results for triangular Rush Hour, again using NCL. In 2016, Solovey and Halperin [14] proved that Rush Hour is also PSPACE-complete with 2×2 cars and immovable 0×0 (point) obstacles.[2] Unlike 1×2 cars, which have an obvious direction of travel (the long direction), 2×2 cars need to have a specified direction, horizontal or vertical.

Back in 2002, Hearn, Demaine, and Tromp [7, 17][3] raised a curious open problem: might 1×1 cars suffice for PSPACE-completeness of Rush Hour? Like 2×2 cars, each 1×1 car has a specified direction, horizontal or vertical. This **1 × 1 *Rush Hour*** problem behaves fundamentally differently: deciding whether a specified car can move at all is polynomial time [7, 8, 17], whereas the analogous questions for 1×2 or 2×2 Rush Hour (or for 1×2 sliding blocks) are PSPACE-complete [7, 8]. Tromp and Cilibrasi [17] exhaustively searched all 1×1 Rush Hour puzzles of a constant size, and found that the length of solutions grew rapidly, suggesting exponential-length solutions; for example, the hardest 6×6 puzzle requires 732

[1] Sadly, to our knowledge, Rush Hour the puzzle was not an inspiration for Rush Hour the 1998 buddy cop film starring Jackie Chan and Chris Tucker.

[2] Solovey and Halperin [14] state their result in terms of unit-square cars amidst polygonal obstacles, but crucially allow the cars to be shifted by half of the square unit. Phrased as cars aligned on a unit grid, these cars are effectively 2×2.

[3] The open problem was first stated in the ICALP 2002 version of [7], based on discussions with John Tromp, as mentioned in [17], which is cited in the journal version of [7].

moves. They also suggested a variant where some cars cannot move at all (perhaps they ran out of gas?), which we call ***fixed blocks*** by analogy with pushing block puzzles [2],[4] as potentially easier to prove hard.

In this paper, we settle the latter open problem by Tromp and Cilibrasi [17] by proving that 1×1 Rush Hour with fixed blocks is PSPACE-complete. This result is the culmination of many efforts to try to resolve this problem since it was posed in 2005; see the Acknowledgments.

Our reduction starts from NCL, and reduces through another related puzzle game, Subway Shuffle. In his 2006 thesis, Hearn [6, 8] introduced this type of puzzle as a generalization of 1×1 Rush Hour, again to help prove it hard. Subway Shuffle involves motion planning of colored tokens on a graph with colored edges, where the player can repeatedly move a token from one vertex along an incident edge of the same color to an empty vertex, and the goal is to move a specified token to a specified vertex. Despite the generalization to graphs and colored tracks, the complexity remained open until 2015, when De Biasi and Ophelders [1] proved it PSPACE-complete by a reduction from NCL. Their proof works even when the graph is planar and uses just four colors.

We use a variant on Subway Shuffle where the graph is directed, and tokens can travel only along forward edges. In Section 3, we prove that directed Subway Shuffle is PSPACE-complete even with planar graphs and just two colors, by a proof similar to that of De Biasi and Ophelders [1]. In Section 4, we then show that this construction uses a limited enough set of vertices that it can actually be embedded in the grid and simulated by 1×1 Rush Hour, proving PSPACE-completeness of the latter with fixed blocks. We conclude with open problems in Section 5.

2 Basics

First we precisely define the problems introduced above.

▶ **Definition 2.1.** *In **Rush Hour**, we are given a square grid containing nonoverlapping **cars**, which are rectangles with a specified orientation, either horizontal or vertical. A legal move is to move a car one square in either direction along its orientation, provided that it remains within the square and does not intersect another car. The goal is for a designated special car to reach the left edge of the board. We also allow **fixed blocks**, which are spaces cars cannot occupy.*

▶ **Definition 2.2.** *1×1 **Rush Hour** is the special case of Rush Hour where each car is 1×1.*

▶ **Definition 2.3.** *In **Subway Shuffle**, we are given a planar undirected graph where each edge is colored and some vertices contain a colored token. A legal move is to move a token across an edge of the same color to an empty vertex. The goal is for a designated special token to reach a designated target vertex.*

▶ **Definition 2.4.** *In **oriented Subway Shuffle**, we are given a planar directed graph where each edge is colored and some vertices contain a colored token. A legal move is to move a token across an edge of the same color, in the direction of the edge, to an empty vertex, and then flip the direction of the edge. The goal is for a designated special token to reach a designated target vertex.*

[4] Tromp and Cilibrasi [17] refer to 1×1 Rush Hour as "Unit (Size) Rush Hour" and the fixed-block variant as "Walled Unit Rush Hour".

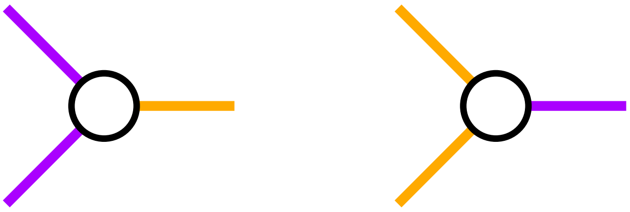


Figure 1 The valid Subway Shuffle vertices with degree 3. Every vertex with degree 1 or 2 is valid.

▶ **Lemma 2.5.** *Subway Shuffle, oriented Subway Shuffle, and Rush Hour are in PSPACE.*

Proof. We can solve these problems in nondeterministic polynomial space by guessing each move, and accepting when the special car or token reaches its goal. So all three problems are contained in NPSPACE, and by Savitch's theorem [12] they are in PSPACE. ◀

3 2-color Oriented Subway Shuffle is PSPACE-complete

In this section, we show that 2-color oriented Subway Shuffle is PSPACE-complete. To do so, we reduce from nondeterministic constraint logic, which is PSPACE-complete [8]. Our reduction is an adaption of the proof in [1] in which the gadgets use only two colors (instead of four) and work in the oriented case.

We actually prove a slightly stronger result in Theorem 3.1: that 2-color oriented Subway Shuffle is PSPACE-complete even with a restricted vertex set, and with a single unoccupied vertex. A vertex is ***valid*** if it has degree at most 3, and has at most 2 edges of a single color attached to it; these vertices are shown in Figure 1. Our proof of PSPACE-hardness will only use valid vertices.

▶ **Theorem 3.1.** *2-color oriented Subway Shuffle with only valid vertices and exactly one unoccupied vertex is PSPACE-complete.*

Proof. Containment in PSPACE is given by Lemma 2.5. To show hardness, we reduce from planar NCL with AND and protected OR vertices.

In constraint logic, a ***protected OR vertex*** is an OR vertex (one with three blue edges) such that two edges, due to global constraints, cannot simultaneously point towards the vertex. NCL is still PSPACE-complete when every OR vertex is protected [8]. Because of this global constraint, there are only five possible states that a protected OR vertex can be in. In particular, there are only four possible transitions between the states of a protected OR vertex. Our OR gadget only allows these four transitions; in particular it does not allow the transition between only the leftmost edge pointing inward and only the rightmost edge pointing inward, which is the defining transition that a protected OR vertex does not have compared to a normal OR vertex.

The Subway Shuffle instance we construct will have only a single empty vertex (other than the target vertex), called the ***bubble***, which moves around the graph opposite the motion of tokens. Our vertex and edge gadgets work by having the bubble enter them, move around a cycle, and then exit at the same vertex. The effect is that each edge in the cycle flips and each token in the cycle moves across one edge, except that one token by the entrance moves twice.

The general structure of the reduction is as follows. First, we choose any rooted spanning tree on the dual graph of the constraint logic graph. This rooted spanning tree will determine the path the bubble takes to get from one vertex or edge to another. For each edge and vertex in the constraint logic graph, we will replace it with a subway shuffle gadget. The constraint logic edges which are part of our spanning tree will have a path for the bubble to cross through them. Each face of the CL graph has paths connecting vertex gadgets and edge gadgets as necessary to allow the bubble to visit each gadget.

When playing the constructed Subway Shuffle instance, the bubble begins at the root of the spanning tree. The bubble can move down the tree by crossing edge gadgets until reaching a desired face. It then enters a vertex or edge gadget, goes around a cycle, and exits. A sequence of moves of this form corresponds to flipping a constraint logic edge or reconfiguring a vertex (that is, changing which constraint logic edge(s) are used to satisfy that vertex and therefore are locked from being flipped away from it). The bubble can always travel back up the spanning tree to the root, and from there visit any face and then any CL vertex or edge.

Now we will describe the various gadgets that implement constraint logic in Subway Shuffle. Many places in the gadget figures have an empty vertex attached to them; this represents where the gadget is connected to the spanning tree. Entering through these vertices is the only way the bubble can interact with a gadget.

The edge gadget is shown in Figure 2. The two vertices and edge at the bottom and top of the edge gadget (in a gray box in the figure) are shared with the connecting vertex gadget. The edge gadget consists of five interlocking cycles. The edge can be flipped by rotating each of the five cycles in order, as shown in Figure 3. The bubble rotates a cycle by entering at the appropriate white vertex, and then moving around the cycle, and finally exiting where it entered.

If the edge is in the spanning tree, we include the rightmost vertex called the ***exit***, which allows the bubble to visit the edge gadget and pass through to face on the other side of the edge. We place the edge gadget in the orientation so that the entrance is on the face closer to the root of the spanning tree of the dual graph.

There are two kinds of edges in constraint logic: red and blue edges. The only difference is that they have different weights for the constraint logic constraints. Blue edges are as shown in Figure 2 (and can be rotated); red edges are the same gadget, but reflected.

Edge gadgets connect to vertex gadgets by sharing the two vertices and edge marked in a gray box. In the edge gadget, when the vertex colors and edge direction are as shown in the edge gadget figure, the edge is ***unlocked***, which means that the bubble is free to flip the direction of that edge. The vertex gadget's colors take precedence for the shared edges and vertices. When they do not match those shown in the edge gadget figure, we say the edge is ***locked***. When this happens, it becomes impossible for the bubble to rotate first cycle, and thus prevents the bubble from flipping the edge. This mechanism is what allows the gadgets to enforce the constraints of the vertices in the constraint logic graph. Edges are only ever locked while pointing into a vertex because all of the constraints in constraint logic only give lower bounds on the number of inward pointing edges. When an edge is pointing away from a vertex, some of the cycles in the vertex will be impossible to rotate, preventing the bubble from unlocking other edges.

The AND vertex gadget is shown in Figure 4. Whenever the bubble is not visiting the vertex gadget, either the blue (weight 2) edge or both red (weight one) edges are locked to point towards the vertex. If all three edges are pointing towards the vertex, the bubble can visit the vertex gadget (at the top entrance) and go around the cycle to switch which edges are locked. This implements the constraints on a NCL AND vertex.

Our protected OR vertex gadget is shown in Figure 5. The two protected edges are the leftmost and rightmost edges, so we can assume that they never both point towards the vertex. The gadget has three entrances. The gadget can be in five possible states corresponding

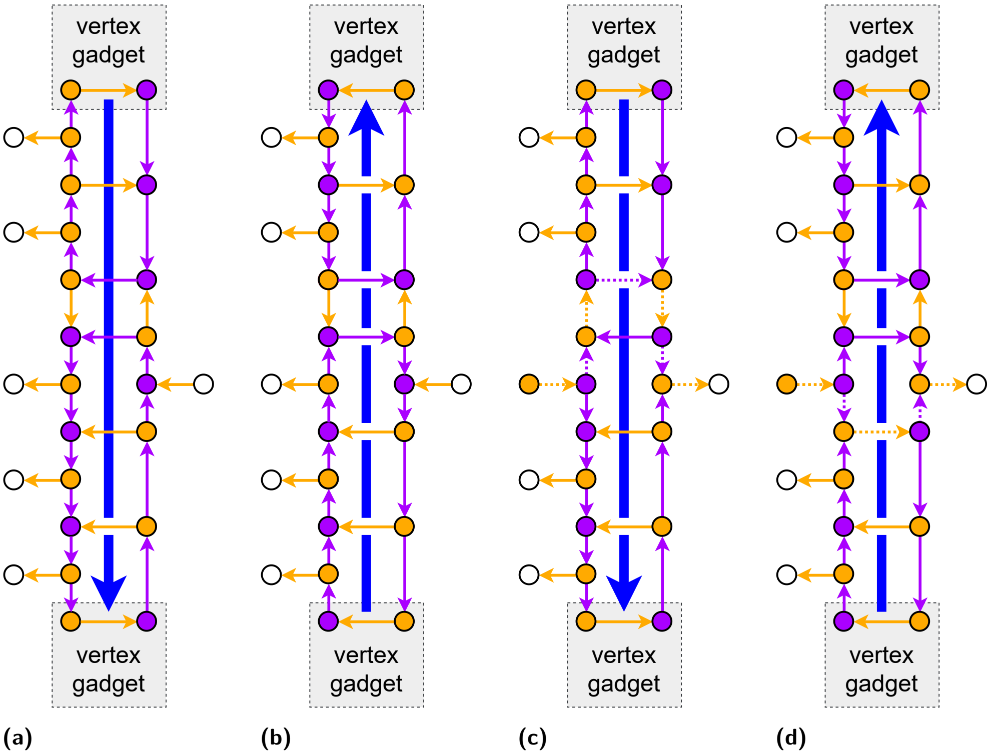


Figure 2 The edge gadget for 2-color oriented Subway Shuffle, shown (a) directed down and unlocked, (b) directed up and unlocked, (c) directed down after the bubble has passed through, and (d) directed up after the bubble has passed through. This gadget is based on the edge gadget in [1].

to the five possible states of a CL protected OR. In each state, the edges which are locked in correspond to the set of edges that are pointing inward in the corresponding state of a CL protected OR. In the first state, the left edge is locked, and the other two are free. In the second state, the middle edge is also locked. In the third state, only the middle edge is locked. In the fourth state, both the middle and right edges are locked. Finally, in the fifth state, only the right edge is locked. To get from one state to the next, the bubble rotates a single cycle. The fives states and the transitions between them are shown in Figure 5. The only transitions between states are to the next and previous states. To transition from one state to the next, the bubble goes around the cycle indicated by the dotted edges.

Our last gadget is the win gadget, shown in Figure 6. It is placed attached to the edge gadget corresponding to the target edge in the constraint logic instance, and allows the player to win the Subway Shuffle instance when that edge can be flipped.

In the first state shown, the target edge is pointing away. If the bubble arrives at the win gadget, it cannot accomplish anything. If the target edge is flipped so it now points toward the win gadget, we will be in the second state. Then the bubble can enter the win gadget at the top entrance and go around the indicated cycle, moving the special token one to the left. Finally, the bubble can enter at the bottom entrance to move the special token across to the target vertex.

To allow the bubble to reach every gadget, we connect the entrances and exits of gadgets which are on the same face of the CL graph. This simply requires a tree connecting these vertices for each face. Each face other than the root of the spanning tree has exactly one edge exit on it; we orient the edges on that face to point towards this exit. The color of these

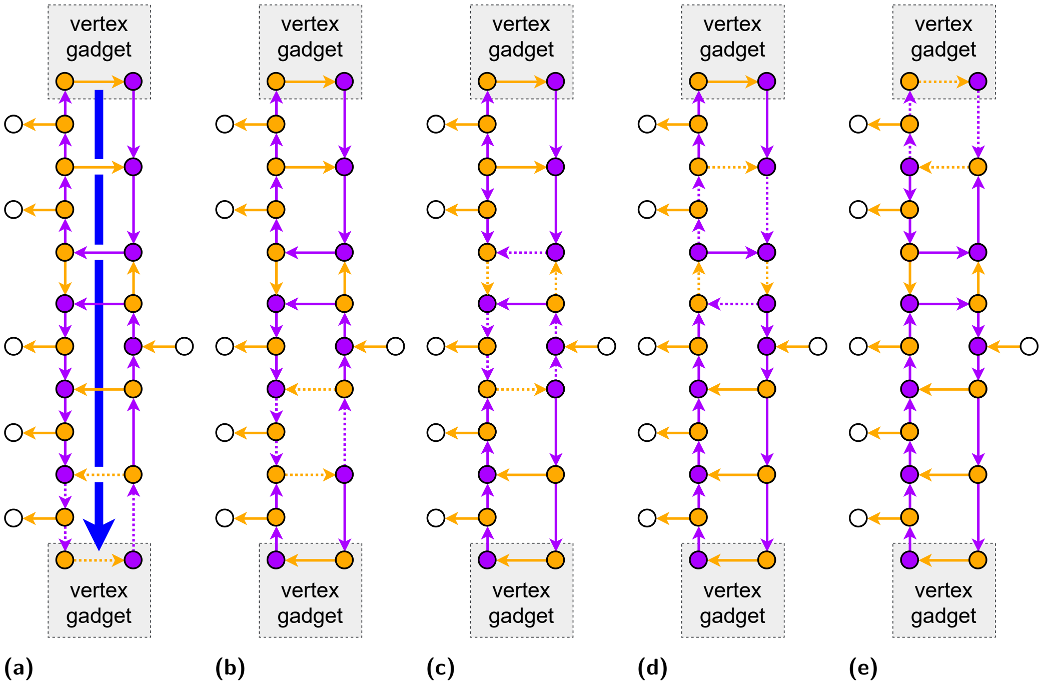
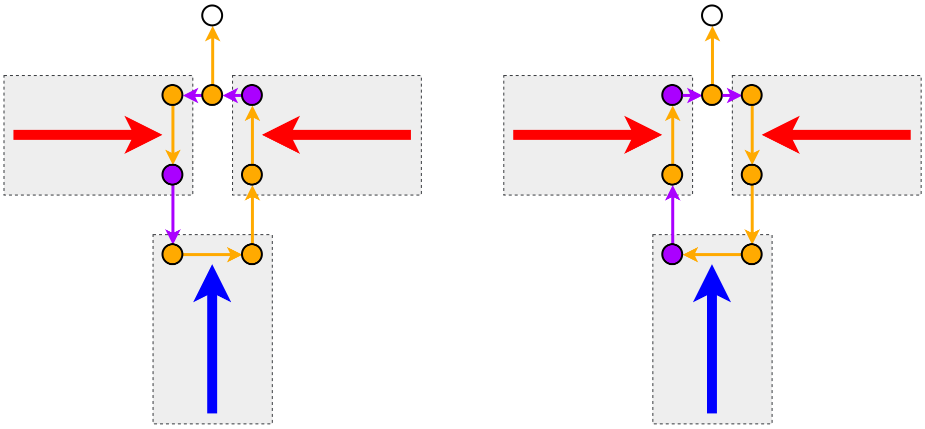


Figure 3 The five cycles that the bubble rotates to flip the orientation of an edge gadget. For each cycle, the bubble enters at the white vertex, goes around the dotted cycle, and leaves where it entered. Note that the colors and orientation of the edge and vertices that connect to the vertex gadget will not always match what is shown in this figure. When they do not, we say the edge is ***locked*** by the corresponding vertex gadget, and it is not possible to rotate the dotted cycle.

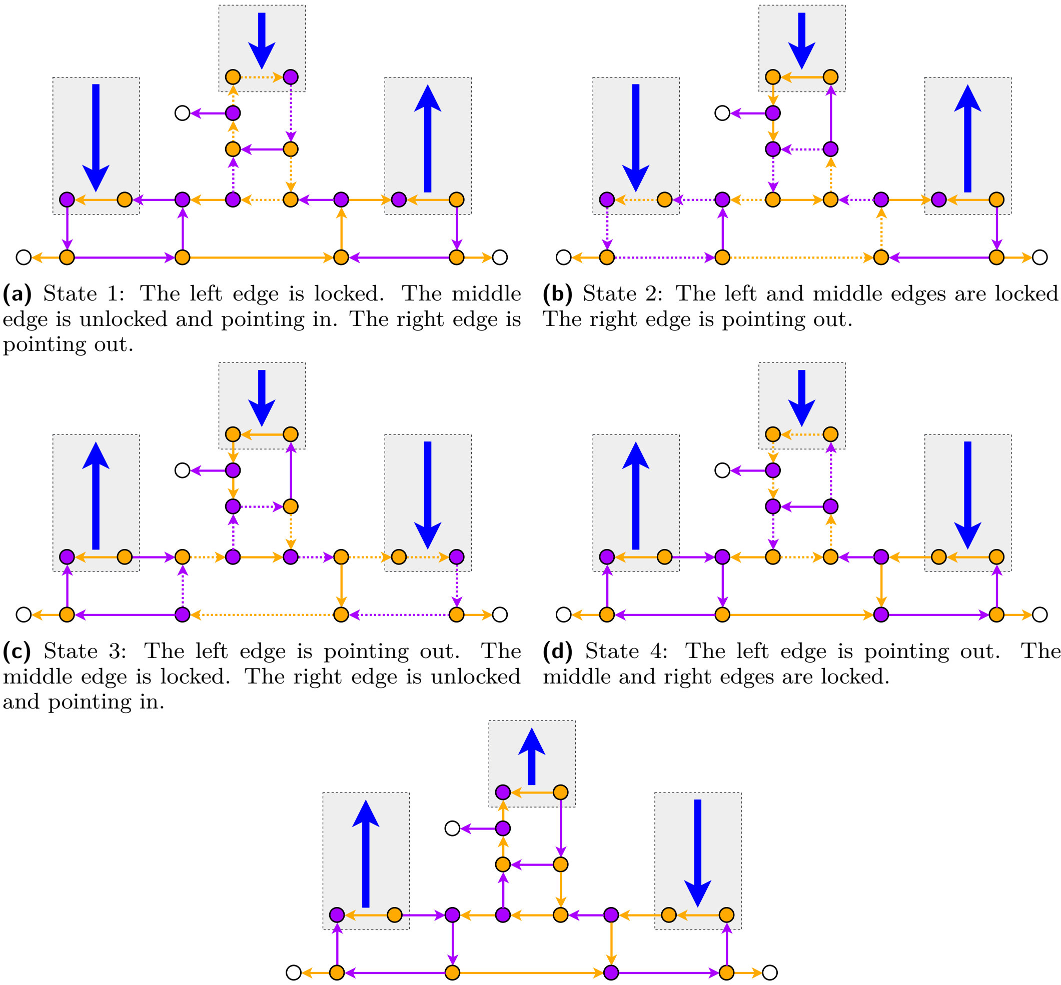


(a) All edges oriented in, with the blue edge locked.

(b) All edges oriented in, with both red edges locked.

Figure 4 The AND vertex gadget for 2-color oriented Subway Shuffle. This gadget is based on the AND vertex gadget in [1].

edges does not matter, provided all vertices are valid and the token at the tail of an edge is the same color. For the face which is the root of the spanning tree, the tree connecting entrances has one vertex without a token, and the edges point towards it; this is where the bubble starts.



(a) State 1: The left edge is locked. The middle edge is unlocked and pointing in. The right edge is pointing out.

(b) State 2: The left and middle edges are locked. The right edge is pointing out.

(c) State 3: The left edge is pointing out. The middle edge is locked. The right edge is unlocked and pointing in.

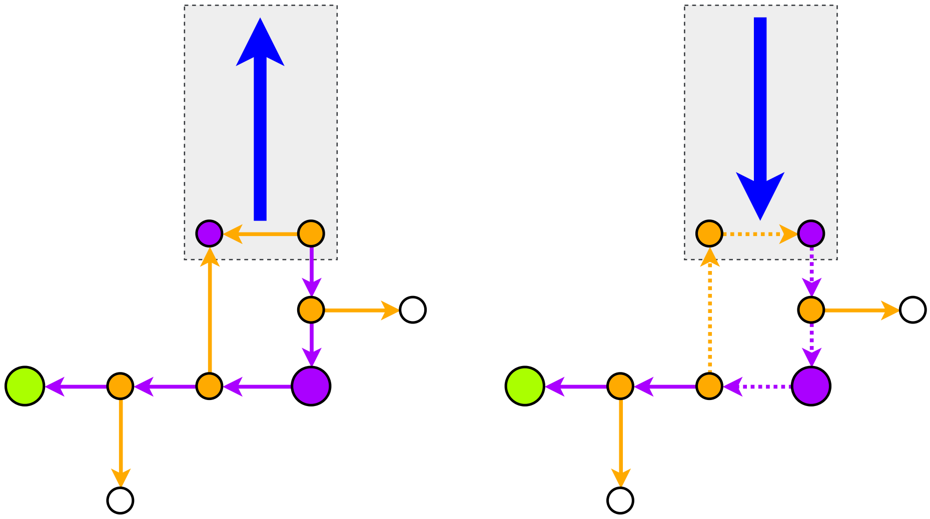
(d) State 4: The left edge is pointing out. The middle and right edges are locked.

(e) State 5: The left and middle edges are pointing out. The right edge is locked.

Figure 5 The five states of the protected OR vertex. The dotted edges show the cycle that is rotated to transition to the next state. Note that each state is defined only by which edged are locked in; the other unlocked edges can be either pointing in or out in each state.

Now we show how the gadgets prevent any moves other than the moves outlined above that simulate the NCL instance. First we consider the edge gadget. It is easy to check that while rotating any of the dotted cycles in an edge gadget, there are only two legal moves other than continuing the cycle. The first one is leaving through the exit vertex during the third cycle. This is equivalent to the bubble just using the throughway in the edge gadget to reach the rest of the spanning tree after turning only the first two cycles. By Lemma 3.3, this is never useful. The other legal move is while turning the first, fourth, or fifth cycle, it is possible for the bubble to move into the connecting vertex gadget through the shared vertices. We will show that nothing useful can be accomplished here when we consider the vertex gadgets. Similarly, it will also be possible for the bubble to come from a vertex and enter the edge gadget through the shared vertices. We show this is not useful in Lemma 3.2.

▶ **Lemma 3.2.** *It is never useful for the bubble to enter an edge gadget directly from a vertex gadget through the shared vertices.*



(a) The locked win gadget. The bubble cannot do anything here.

(b) The unlocked win gadget. Now that the blue edge is pointing into the gadget, the indicated cycle can be rotated.

Figure 6 The win gadget for 2-color oriented Subway Shuffle. The target edge starts pointing away. If it is flipped, the bubble can enter the win gadget once at each entrance to move the (bottom right) purple special token to the (bottom left, green) target vertex. This gadget is based on the FINAL gadget in [1].

Proof. We need to check the up, down, up traversed, and down traversed configurations.

In most configurations, there are no legal moves to enter the edge gadget from the shared vertices. The only configuration where this is possible is from the orange token on the top left of the upward pointing edge gadget. From here, it can move through a path of three tokens before it gets stuck. At that point, the only legal move is to undo the last three moves and exit the same way it entered. ◀

▶ **Lemma 3.3.** *It is never useful to turn some of the cycles in an edge gadget without turning all of them.*

Proof. If you turn some of the cycles, but not all of them, then both ends of the edge gadget will be in the pointing outward configuration. For all of the vertex gadgets, there are no transitions that require the outward pointing configuration, so the edge gadget being in this configuration never lets you make a move that you could not make if you finished turning all of the cycles in an edge gadget.

We also need to make sure that turning only some cycles, and then entering an edge gadget from a vertex gadget (as in Lemma 3.2), does not allow you to do anything. If we look at all of the partial edge configurations as shown in Figure 3, there is no way to access anything from any of these configurations. We also need to check the configurations that arise from partially rotating an edge and then traversing it. Since it is not possible to reach the traverse paths from entering from a vertex gadget, these configurations also do not let the bubble do anything else useful. ◀

Now we consider the AND gadget. Since the entire gadget is a single cycle, there is nothing the bubble can do within the gadget while turning the cycle. While turning the cycle, the bubble can try to enter an edge gadget through one of the shared vertices; however, we have already shown that the this is never useful in Lemma 3.2.

We also need to consider if the bubble enters the vertex gadget from an edge on one of the shared vertices. It will never be able to move around the entire cycle because the orange vertex at the top will not be accessible. The only other thing the bubble can do is try to enter a different edge gadget, but we already showed this is not useful in Lemma 3.2.

Now we consider the OR gadget. First we look at each of the four cycles. While turning the first cycle, the only legal move that is not continuing the cycle is moving the purple token just to the right of the cycle. However, from here, the only moves lead to dead ends so there is not anything useful for the bubble to do besides immediately return to the cycle. There are no other legal moves while turning the second cycle. While rotating the third cycle, it is possible for the bubble to reach the shared vertices of the leftmost edge gadget, but by Lemma 3.2 this does not help. While rotating the fourth cycle, it is possible for the bubble to reach the shared vertices of the rightmost edge gadget, but again this does not help.

Now we consider when the bubble enters the OR vertex gadget from an edge gadget through one of the shared vertices. In the first state, there are no legal moves after entering from the top or right edges. In the second state, from either the top or left edges it can enter and traverse most of the gadget but cannot complete any loop and thus cannot make progress by Lemma 3.4. In the third state, the bubble has no legal moves after entering from the left or top edge. From the right edge it can traverse most of the gadget but cannot complete any loops. From the fourth state, again, while the bubble can traverse most of the gadget after entering from the top edge, it does not complete any loops so it has no effect. In the fifth state, there are no legal moves after entering the vertex gadget.

▶ **Lemma 3.4.** *If the bubble takes any path from any vertex to the same vertex which does not complete a nontrivial loop, then the state of the Subway Shuffle instance must not have changed.*

Proof. If the bubble never completed a loop, then the only way for it to get back to where it started is to take the same path in reverse. By the definition of Subway Shuffle moves, this exactly undoes these moves returning the instance back to its original state. ◀

Finally, we check the win gadget. While using the win gadget, there are no legal moves other than completing the one loop. There is only one edge connected to the win gadget. If the bubble tries to enter the win gadget here, it cannot leave anywhere else or complete any loops, so by Lemma 3.4 it must return with no effect.

Since the constraint logic graph is planar, the reduction yields a planar graph for 2-color oriented Subway Shuffle. Since the constructed instance Subway Shuffle is winnable exactly when the constraint logic instance is, and the reduction can clearly be done in polynomial time, this shows 2-color oriented Subway Shuffle is PSPACE-hard. All of the gadgets used, including the trees connected gadget entrances, use only valid vertices, so it is still PSPACE-hard with only valid vertices. ◀

4 1 × 1 Rush Hour is PSPACE-complete

In this section, we show that 1 × 1 Rush Hour is PSPACE-complete by a reduction from 2-color oriented Subway Shuffle with only valid vertices and only a single empty vertex, which was shown to be PSPACE-complete in the previous section. 1 × 1 Rush Hour is played on a large square grid. We allow for fixed blocks, which are spaces marked impassable in the grid.

We will simulate Subway Shuffle vertices with individual cars at intersections, and edges as paths of cars. In general, purple edges and vertices will be horizontal cars, and orange edges and vertices will be vertical cars. Like in the Subway Shuffle, we will have a single ***bubble*** which is a single empty space that moves around as cars move into that space.

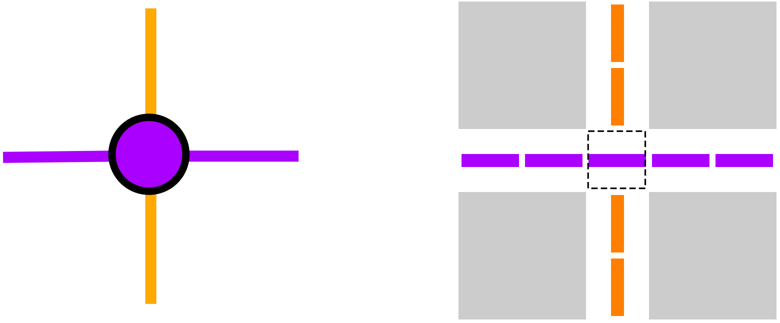


Figure 7 A degree-4 Subway Shuffle vertex embedded in Rush Hour. Note that, while this is not a valid Subway Shuffle vertex, all valid vertices are subsets of this vertex. Individual dashes represent cars. A line of cars of one color represents a Subway Shuffle edge of that color. The center boxed car represents the Subway Shuffle vertex.

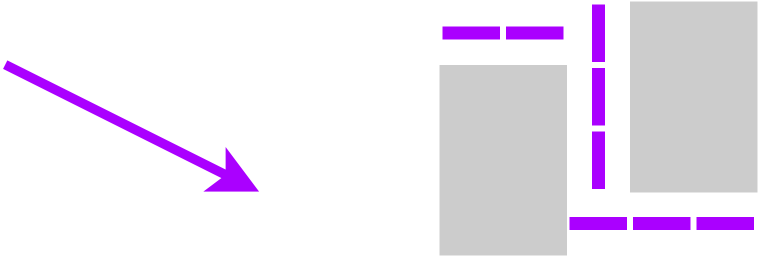
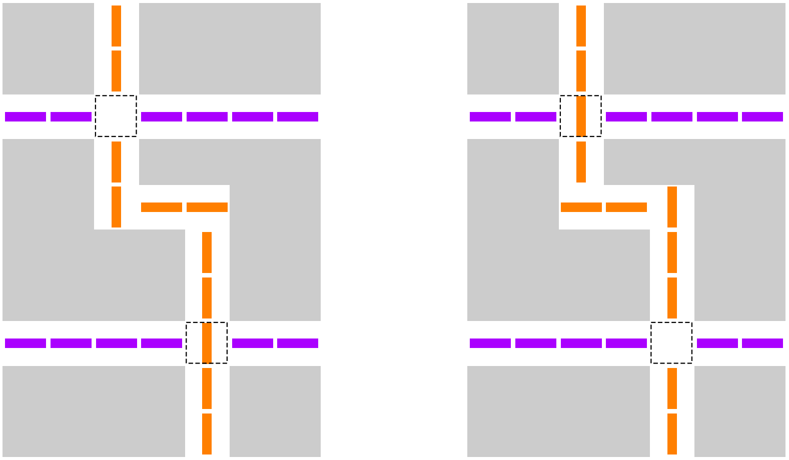


Figure 8 A Rush Hour simulation of a Subway Shuffle edge. This is a purple edge which points right.

We replace each vertex in our Subway Shuffle instance with a single car which is vertical if there is an orange token there, and horizontal if a purple token is there. Orange edges leading from a vertex attach to it as vertical rows of cars, and purple edges attach to a vertex as horizontal rows of cars. A degree-4 vertex with a purple token is depicted in Figure 7. Valid vertices can be embedded this way, with fixed blocks on the unused sides for lower degree vertices.

A Subway Shuffle edge is simulated by a path of cars which can make right-angle turns, allowing us to embed an arbitrary planar Subway Shuffle graph. The direction of a car at a turn in an edge defines which way the Subway Shuffle edge is oriented. A purple edge which points right is depicted in Figure 8. In order to maintain the directionality of edges, each edge must be simulated by a path with at least one turn.

To make a move, suppose the bubble is currently at a vertex. To move a token in from an adjacent vertex, a car from the connecting edge is moved in. Then cars from that edge are all moved one space toward the initial vertex, until finally we can move the car in the second vertex out. Note that this process reverses the orientation of the edge as desired. If the edge was pointed in the correct direction, then this process will succeed; if the edge is oriented in the wrong direction, then this process will fail when we try to turn a corner in the edge. Similarly it is impossible to move a token along an edge of the opposite color, because it will be unable to move out of its vertex. An example of a single Subway Shuffle move where an orange token is moved up along an orange edge embedded in Rush Hour is shown in Figure 9.



(a) Before moving the orange token up.

(b) After moving the orange token up.

Figure 9 Moving a single orange token in Subway Shuffle when simulated by Rush Hour in Figure 8.

No other useful actions can be taken. If the bubble is not currently at a vertex, then there are at most two possible moves. One of them would just be undoing the previous move, and the other would be continuing the process of moving a token along an edge. When the bubble is at a vertex, moving any adjacent car into the vertex is the same as starting the process of moving a Subway Shuffle token along the corresponding edge.

The win condition of a Rush Hour instance is allowing the marked car to escape the grid. The win gadget needs to be specified more precisely because Subway Shuffle tokens do not correspond exactly to Rush Hour cars. Also, we want to make sure that everything can fit within a grid so our win condition is actually located near the edge.

Our win gadget is depicted in Figure 10. The win condition is the circled car reaching the star. The boxed cars represent Subway Shuffle vertices. In order to win, first the boxed orange car directly in front of the circle car must leave by rotating this cycle. This represents the marked token in the Subway Shuffle vertex moving to the middle vertex along the bottom of the win gadget. Then, the leftmost orange line must be moved down one space, clearing the way for the marked car to leave.

In Rush Hour, because winning requires a car leaving the grid, we must also take care to make sure that the win gadget is at the boudary of our construction, and not somewhere buried in the middle. To do this, we conisder the CL edge which is part of the win gadget. Since the CL graph is planar, we can consider one of the faces that this edge is a part of, and make this face the "outside" face. Now our win gadget is at the boundary, which is what we needed.

5 Open Problems

In this paper, we have shown that 1 × 1 Rush Hour with fixed blocks is PSPACE-complete, solving Tromp and Cilibrasi's open problem [17]. It remains whether the assumption of fixed blocks can be eliminated, and thereby solve the open problem of Hearn, Demaine, and Tromp [7, 17]. We note that it is impossible to perfectly simulate a fixed block using Rush Hour cars, since for any arrangement of cars in a region, there must be at least one point along the

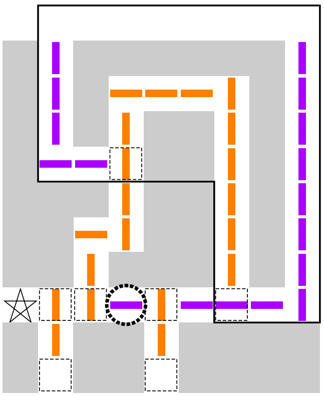


Figure 10 The cycle of the subway shuffle win gadget embedded in Rush Hour. The goal is to get the circled car to the star. The boxed cars are the vertices in the Subway Shuffle win gadget. The two lines of purple cars extending upward are the purple edges of the connected edge gadget. Everything inside the solid black line is part of the connecting edge gadget.

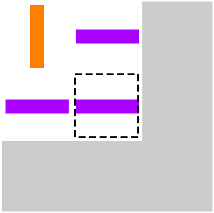


Figure 11 Let the gray area be accessible by the bubble. Then the boxed car is at the corner of the boundary of the accessible region, and regardless of its orientation it must also be accessible by the the bubble.

boundary of the region that, if it were empty, a car can exit the region. For a single bubble, it gets worse than that. Let a space be ***accessible*** if the bubble can ever reach that space. By Theorem 5.1, the accessible region is always a rectangle. Since we can ignore anything inaccessible, we can just assume that everywhere in the entire Rush Hour grid is accessible. Because the bubble can get everywhere, it seems impossible to modify the gadgets in our proof in any simple way to constrain the bubble from wandering freely inside and between the cycles in gadgets.

▶ **Theorem 5.1.** *In any* 1×1 *Rush Hour instance with no fixed blocks with only a single "bubble," the set of accessible spaces is a rectangle.*

Proof. The accessible region is clearly connected. If it is not a rectangle, there must be a corner on the boundary of the accessible region where two accessible spaces are adjacent to the same inaccessible space, as in Figure 11. Then regardless of its orientation, the car in this inaccessible space must be able to move into one of these two accessible spaces, and thus is also accessible. This is a contradiction, so the accessible region must be a rectangle. ◀

References

1 Marzio De Biasi and Tim Ophelders. Subway Shuffle is PSPACE-complete. Manuscript, February 2015. URL: `http://www.nearly42.org/cstheory/subway-shuffle-is-pspace-complete/`.

2 Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann. Push-2-F is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry (CCCG 2002)*, pages 31–35, Lethbridge, Alberta, Canada, August 12–14 2002.

3 Erik D. Demaine and Mikhail Rudoy. A simple proof that the $(n^2 - 1)$-puzzle is hard. *Theoretical Computer Science*, 732:80–84, July 2018.

4 Gary William Flake and Eric B. Baum. Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1–2):895–911, 2002.

5 Martin Gardner. Sliding-block puzzles. In *Martin Gardner's Sixth Book of Mathematical Diversions from Scientific American*. W. H. Freeman and Company, 1971. Republished by MAA, 2001.

6 Robert A. Hearn. *Games, Puzzles, and Computation*. PhD thesis, Massachusetts Institute of Technology, 2006. URL: `http://erikdemaine.org/theses/bhearn.pdf`.

7 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, October 2005. Originally appeared at ICALP 2002.

8 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. AK Peters/CRC Press, 2009.

9 Patentarcade.com. Case update: Rubin v. Apple Inc. Blog post, 7 July 2011. URL: `http://patentarcade.com/2011/07/new-case-rubin-v-apple-inc.html`.

10 Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, 1990. URL: `http://users.soe.ucsc.edu/~manfred/pubs/J15.pdf`.

11 Don Rubin. The Parking Lot. `http://www.donrubin.com/parking_lot.html`, 2012.

12 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. `doi:10.1016/S0022-0000(70)80006-X`.

13 Jerry Slocum and Dic Sonneveld. *The 15 Puzzle*. Slocum Puzzle Foundation, 2006.

14 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *The International Journal of Robotics Research*, 35(14):1750–1759, November 2016. `doi:10.1177/0278364916672311`.

15 James A. Storer. Tokyo Parking / Rush Hour. Jim Storer Puzzles Home Page, 2015. URL: `https://www.cs.brandeis.edu/~storer/JimPuzzles/ZPAGES/zzzTokyoParking.html`.

16 ThinkFun. The evolution of ThinkFun's Rush Hour. Blog post, February 2018. URL: `http://info.thinkfun.com/stem-education/the-evolution-of-thinkfuns-rush-hour`.

17 John Tromp and Rudi Cilibrasi. Limits of Rush Hour Logic complexity. *arXiv preprint cs/0502068*, 2005. URL: `https://arXiv.org/abs/cs/0502068`.

18 Stephen A. Wagner. Manipulable puzzle. U.S. Patent D395,468, June 1998. URL: `https://patents.google.com/patent/USD395468S/en?oq=d395468`.

An Optimal Algorithm for Online Freeze-Tag

Josh Brunner
Massachusetts Institute of Technology, Cambridge, MA, USA
brunnerj@mit.edu

Julian Wellman
Massachusetts Institute of Technology, Cambridge, MA, USA
wellman@mit.edu

Abstract

In the *freeze-tag problem*, one active robot must wake up many frozen robots. The robots are considered as points in a metric space, where active robots move at a constant rate and activate other robots by visiting them. In the (time-dependent) *online* variant of the problem, each frozen robot is not revealed until a specified time. Hammar, Nilsson, and Persson have shown that no online algorithm can achieve a competitive ratio better than 7/3 for online freeze-tag, and posed the question of whether an $O(1)$-competitive algorithm exists. We provide a $(1+\sqrt{2})$-competitive algorithm for online time-dependent freeze-tag, and show that this is the best possible: there does not exist an algorithm which achieves a lower competitive ratio on every metric space.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithm, competitive ratio, freeze-tag

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.8

Acknowledgements The generalization of the metric in Section 4 to the construction of Section 5 was inspired by a discussion with Yuan Yao. We thank David Karger and Aleksander Madry for teaching the algorithms course where this project began, and for their useful feedback on a draft of this paper. We also thank three peer editors, Leo Castro, Roberto Ortiz, and Tianyi Zeng for their helpful comments on an early draft. Finally, we appreciate the helpful comments of the anonymous reviewers of this paper.

1 Introduction

In the freeze-tag problem (FTP), there are n robots, represented by points in a metric space. Each robot is either awake (active) or asleep (frozen), and initially only one is awake. The goal is to get all the active robots to wake up all the asleep robots in the minimum possible time. Only the active robots may move, and whenever they reach an asleep robot, that robot wakes up and can now help wake up additional robots. All active robots move at the same constant rate. A solution to the problem consists of a route which wakes up all of the robots, and is optimal if it wakes up all the robots in the minimal possible time.

The freeze-tag problem can be interpreted as finding a minimum-depth directed spanning tree on a set of points, where each vertex has out-degree at most two [1]. The first work on the problem was done in this language, e.g. in [4], and was motivated by (for example) the IP multicast problem, where a server needs to distribute information to a set of hosts. The freeze-tag problem was introduced under this new name in [1], and finding an optimal solution was shown to be NP-hard. Further work has mostly centered around approximation algorithms such as in [2] and [6]. No PTAS for general metric spaces has been found, though much progress has been made for Euclidean metrics in [5] and [7], finding a linear-time PTAS in some cases. In [1] it is shown that even 5/3-approximation is NP-hard for general metrics arising from weighted graphs, so assuming $P \neq NP$, no PTAS exists.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 8; pp. 8:1–8:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we focus on the online version of this problem, where the asleep robots, or *requests*, are not known in advance. Each request is released at a certain time, before which the location, time, or existence of the request is not known. The goal is still to minimize the time when the last asleep robot is reached. This variant was named *time-dependent* freeze-tag (TDFT) by Hammar, Nilsson, and Persson [3]. The online problem models the schoolyard game of freeze-tag more closely, since one doesn't know where or when the next person will get tagged. We feel that it may also be more relevant for some applications, where requests for information may be unpredictable.

Hammar, Nilsson, and Persson are concerned with the *competitive ratio* achieved by an online algorithm, to model the worst-case performance. They show that no algorithm can achieve a competitive ratio lower than 7/3, by giving a specific metric where this is not possible [3, Theorem 5]. They ask whether there is any online algorithm that achieves a constant competitive ratio. We will slightly improve their bound (through a generalized construction) to show that no competitive ratio lower than $1+\sqrt{2}$ is possible, and give an algorithm which achieves this ratio.

2 Setup and Results

Recall that a metric space M is a set equipped with a distance function $d : M \times M \to \mathbb{R}$ which is non-negative, symmetric, and satisfies the triangle inequality. Metric spaces induce a topology generated by the open balls $B_\varepsilon(x) := \{y \in M : d(x,y) < \varepsilon\}$, for $\varepsilon > 0$ and $x \in M$. Examples of metric spaces include those arising graphs with weights satisfying the triangle inequality, or, say, Euclidean spaces. For time-dependent freeze-tag, it only makes sense to use a special class of metric spaces.

▶ **Definition 2.1.** *A metric space (M,d) is* strongly connected *if for any two points $x, y \in M$, there exists a continuous function $f : [0,1] \to M$ with endpoints $f(0) = x$ and $f(1) = y$, and also for each $z \in (0,1)$, we have that $d(x, f(z)) + d(f(z), y) = d(x,y)$.*

Intuitively, a metric space is strongly connected if it is connected and for all $x, y \in M$ there is a "shortest" path from x to y which is also a shortest path to each point along the path. If a connected metric space (M,d) is not strongly connected, we can instead use the metric space (M,d'), where for all $x, y \in M$, we have

$$d'(x,y) := \min_{P:x\to y} \max_{z\in P} d(x,z) + d(z,y).$$

The metric (M,d') is strongly connected, because for every point z on the minimal path P from x to y, the sum of its distances to x and y is at most $d'(x,y)$. The paths P our robots can take will be parameterized by the time t, and must satisfy $|d(P(t),x) - d(P(t'),x)| \le |t - t'|$ for all times t, t' and points $x \in M$. In strongly connected metrics, the distance function actually reflects the time required for a robot to move between points, which is why we make this restriction.

An instance of the freeze-tag problem consists of a list of the n positions $p_0, \ldots, p_{n-1}$ of the robots in a metric space M. The point p_0 denotes the starting position of the one active robot, while $p_1, \ldots, p_{n-1}$ are the positions of the frozen robots. We refer to the robot which began at position p_i by r_i. Solutions to the freeze-tag problem can be considered as binary trees rooted at p_0 which span the positions p_i, where the edges represent robot paths. Then the FTP is equivalent to finding the spanning binary tree rooted at p_0 which has the minimal possible weighted depth.

An instance of the online (time-dependent) problem consists of the same points p_i, but with associated release times t_i. We assume that $0 = t_0 \leq t_1 \leq \cdots \leq t_n$. The offline (normal FTP) problem is the special case where $t_i = 0$ for all i. The robots are denoted $r_i = (p_i, t_i)$. Time-dependent freeze tag necessarily takes place in a strongly connected metric space, so that robots can take paths between any two points x, y, which take $d(x, y)$ time to complete. We do not really lose any generality by restricting to strongly connected spaces, because as noted above any connected space can be modified to be strongly connected, and moreover, it doesn't make much sense to play freeze-tag on spaces which aren't strongly connected in the first place.

A solution to an instance of the TDFT problem consists of a collection of paths in the metric space which unfreezes each robot, while an algorithm for TDFT gives a strategy which says how to move the active robots in any instance of the problem, possibly depending on the metric. An optimal solution to an instance consists of a *optimal scheduling tree*, which unfreezes each robot no earlier than it is released, and minimizes the time which the last robot is unfrozen. The problem of finding the optimal scheduling tree for a given input is NP-hard [1], but it is at least computable. We seek to minimize the *competitive ratio* of an algorithm A for time-dependent freeze-tag. For each instance σ for TDFT, there is an associated time required for the optimal scheduling tree, denoted $OPT(\sigma)$, and a time which the algorithm's solution takes, $A(\sigma)$. We want to minimize the competitive ratio, defined to be $R := \max_{\sigma} \frac{A(\sigma)}{OPT(\sigma)}$.

In [3], Hammar et. al. give an example of a metric space where no algorithm can achieve a competitive ratio lower than 7/3 for the online time-dependent freeze-tag problem. They pose the question of whether there is any algorithm which achieves a constant competitive ratio in every metric space. We answer this question affirmatively, by giving an algorithm which we will show achieves the best possible competitive ratio.

▶ **Theorem 2.2.** *The algorithm described in Section 3 is $(1+\sqrt{2})$-competitive for the online TDFT problem on every continuous metric space. Moreover, for every $\varepsilon > 0$, there exists a continuous metric space where no deterministic algorithm is $(1+\sqrt{2}-\varepsilon)$-competitive for the online TDFT problem.*

In Section 3 we describe our algorithm and show it is $(1+\sqrt{2})$-competitive. In Section 4, we describe a metric space which is extremely similar to the one presented in [3], and use it as an example of type of analysis we will do. While the lower bound derived in this section is actually less than 7/3, it serves to motivate the framework for our more complicated analysis. In Section 5 we generalize the construction, giving an infinite family of metrics (with the metric in the previous section as a base case), and show that the metrics in this family give lower bounds on the competitive ratio that can be arbitrarily close to $(1+\sqrt{2})$, completing the proof of Theorem 2.2.

3 $(1+\sqrt{2})$-Competitive Algorithm

The key idea of our algorithm is *patience*; we hope to have the robots wait near their starting positions until all of the robots are released, at which point we can copy the optimal scheduling tree. Ideally, we don't move any robots until a time t such that the optimal scheduling tree for the current input sequence would take time at most $t/\sqrt{2}$, at which point we use this scheduling tree to wake up all of the robots, taking a total time of $t(1+\frac{1}{\sqrt{2}})$, achieving the desired competitive ratio. Since we do not know the ultimate number of robots which will be released, we cannot know when we truly need to start waking up robots, and so the algorithm needs to be a little fancier. Let's describe our algorithm more precisely now.

Let $OPT(j)$ denote the minimum possible run time of a scheduling tree starting at p_0 which wakes up all of the robots $r_i = (p_i, t_i)$ for $i \leq j$, under the condition that robot r_i is not activated until at least time t_i. Equivalently, $OPT(j)$ is the time of the last unfreezing in the optimal scheduling tree for the instance truncated at r_j. Every time a robot is released, we recompute the value $OPT(j)$. While, as shown in [1], this computation may be NP-hard, it is certainly still computable.

Our online algorithm always has a schedule in mind for waking up the swarm. At every moment, all active robots follow the current schedule, by moving along the shortest paths to their next destination (given by the condition that the metric is strongly connected), at a rate of one distance unit per time unit. Every time t_j when a robot r_j is released, we compute $OPT(j)$ and then overwrite the current schedule with a new schedule (steps described below), and start over from Step 1.

1. Send every active robot r_i back to its starting position p_i.
2. Wait until time $t = \sqrt{2} \cdot OPT(j)$.
3. Wake up the swarm in time $OPT(j)$ by following an optimal schedule.
4. Send every robot r_i back to its starting position p_i, and wait there.

This algorithm appears to be $(1 + \sqrt{2})$-competitive, since it should complete waking up the swarm at time $\sqrt{2} \cdot OPT(j) + OPT(j) = (1 + \sqrt{2}) \cdot OPT(j)$. The main thing which remains to be shown is that the algorithm always completes the first step before time $t = \sqrt{2} \cdot OPT(j)$.

▶ **Lemma 1.** *Under the algorithm described, at any time T, each robot r_i is at a distance at most $\frac{T}{1+\sqrt{2}}$ from its starting position p_i.*

Proof. We note that it suffices to only consider times during Step 3, since at any other time the robots are either at home or moving toward home.

Step 3 started at time at least $\sqrt{2} \cdot OPT(j)$, so if we have been in Step 3 for a duration d, the current time is at least $T \geq d + \sqrt{2} \cdot OPT(j)$. Step 3 takes a total of $OPT(j)$ time, so we also have $OPT(j) \geq d$. Therefore $T \geq d + \sqrt{2} \cdot d$, and so $d \leq \frac{T}{1+\sqrt{2}}$. Since robots move at unit speed, if a robot has been moving for at most d time since the last time it was at its starting position, it must be within d distance of its starting position. Since Step 3 always begins with all robots at the starting position, it follows that each robot is always within $\frac{T}{1+\sqrt{2}}$ of its starting position. ◀

We are now prepared to prove the first half of our main result.

▶ **Theorem 3.1.** *The algorithm described is $(1 + \sqrt{2})$-competitive under any metric for the time-dependent online freeze-tag problem.*

Proof. Let OPT be the time that an optimal schedule would need to wake up all of the robots. Note that if the final request is released at a time t, then $OPT \geq t$, since it is not possible to satisfy a request before it is released. Therefore $t_j \leq OPT$ for all j.

Consider the time $\sqrt{2} \cdot OPT$. Since the last time we received a request was at the latest at time OPT, we have not modified the schedule since then. Thus, we have had at least $(\sqrt{2} - 1) \cdot OPT = \frac{OPT}{1+\sqrt{2}}$ time to complete Step 1 of the algorithm. By Lemma 1, at time OPT, each robot is at most $\frac{OPT}{1+\sqrt{2}}$ away from its starting location. Thus, by time $\sqrt{2} \cdot OPT$, Step 1 will have finished, and all the robots will be at their starting locations. Then Step 3 will begin at time $\sqrt{2} \cdot OPT$, and take at most OPT time, for a total time of $(1 + \sqrt{2}) \cdot OPT$ for when the last robot is awakened. ◀

▶ Remark 3.2. Our algorithm balances staying close to home with having to wait before executing an optimal schedule. Any algorithm for which Lemma 1 holds for a smaller constant than $\frac{1}{1+\sqrt{2}}$ will necessarily have a larger competitive ratio. If there were an algorithm with a better competitive ratio, it seems that it must be willing to send a robot farther away, while keeping others slightly closer. Our proof in the next two sections can be viewed as giving a formal justification for why it isn't viable to keep most of the robots closer to home.

One drawback of our algorithm is that computing $OPT(j)$ takes an exponential amount of time, and so it is not particularly efficient in that sense. One could use an approximation algorithm for $OPT(j)$, which would increase the competitive ratio by the approximation factor. However, even 5/3-approximation is NP-hard, and the best known polynomial-time algorithm for general metrics is only a $(\log n)$-approximation [1], so more work will be required for our algorithm to be executable efficiently.

4 Example Lower Bound Construction

In this section, we use a metric which is very similar to the metric described in [3] which gave a lower bound of 7/3. This metric can give a lower bound on the optimal competitive ratio of $\frac{\sqrt{33}-1}{2} \approx 2.37228\ldots$ by optimizing Lemma 2, which would by itself be an improvement on the 7/3 bound. We will present a simplified analysis which only gives a lower bound of $3\sqrt{2}-2 \approx 2.24264\ldots$, but which aligns better with the methods in Section 5. The complicated family of constructions there can be viewed as generalizing the metric used here, and the analysis will follow a similar strategy. In fact, the metric in this section and the lemmas proved will serve as the base case for an induction argument. We also will use Lemma 4 as a framework for proving stronger bounds.

A lower bound of 2 is easily achieved by not revealing where a single frozen robot is until a time equal to its distance from p_0, adversarially placing it opposite whatever direction r_0 had been moving before that time. To improve on this, Hammar et. al. [3] force two robots to move in one direction, then travel all the way back the other way to unfreeze the final robot. Roughly speaking, the improvement in our approach comes from forcing the robots to move a little bit farther before turning back.

Our metric space M is formed by a weighted graph with 8 vertices. The origin p_0 is where the initial active robot starts, and there will be 7 other points $p_1, \ldots, p_7$, which aren't necessarily starting points for robots. We place $p_1, \ldots, p_6$ on edges at distance 1 from the origin, while p_7 is at a distance $r := 1+\sqrt{2}$ from the origin. We also add edges of length one connecting points $(p_1, p_2), (p_3, p_4)$, and (p_5, p_6), forming three equilateral triangles with the origin. The metric consists of all points along edges in this graph, with distances given by shortest path between the points in the graph, and is pictured in the Figure 1 below.

The construction in [3] has exactly the same structure, but with different edge lengths. Note that any connected undirected graph with weights satisfying the triangle inequality naturally gives rise to a strongly connected metric space, with distances equal to the lengths of the shortest paths in the graphs. The points in the metric include all points along the edges of the graph.

We can now start describing a TDFT instance for this metric. First fix an algorithm A on the metric M. Let $r_0 = (p_0, 0)$ and also $r_1 = (p_0, 0)$, so that we will have $N := 2$ active robots available from the start. Do not release any other robots until time 1. At time $t = 1$, there must a triangle where neither of the robots are. Since the triangles are all the same, without loss of generality we will assume that this is the triangle $p_0p_3p_4$. Then release one robot each at p_3 and p_4 at time $t = 1$. In summary, we define the input

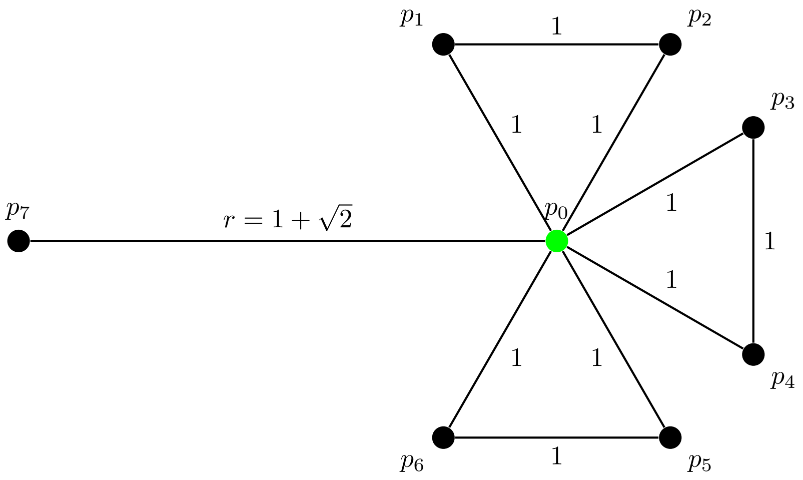


Figure 1 The metric space M.

$\sigma_A \coloneqq (p_0, 0), (p_0, 0), (p_3, 1), (p_4, 1)$. Certainly the time $A(\sigma_A)$ which the algorithm takes to unfreeze both frozen robots is at least 2, since both robots are at least one away from both frozen robots at $t = 1$. The following lemma will be key for our analysis.

▶ **Lemma 2.** *For any online algorithm A on the metric M with input σ_A, we have either $A(\sigma_A) \geq 3\sqrt{2} - 2 =: R$, or there is some time $1 + \sqrt{2} \geq t \geq 2$ such that all but at most $N - 2 \coloneqq 0$ robots are at a distance more than $t(\sqrt{2} - 1)$ from p_0, and closer to a frozen robot than p_0 is.*

Proof. Suppose at every time $t \in [2, 1 + \sqrt{2}]$ there is a robot within $t(\sqrt{2} - 1)$ of the origin. The earliest time that either of the initial requests can be satisfied is time 2. Without loss of generality, let robot r_0 satisfy the request at p_3. At the point when r_0 reaches p_3, the only way to finish before time $t = 3$ is if the request at p_4 is satisfied by our other initial robot r_1. When r_1 reaches p_4, it is at a distance 1 from p_0, which is greater than $t(\sqrt{2} - 1)$ when $t < 1 + \sqrt{2}$. Then the lemma will hold if r_1 is closest to p_0, so assume some robot goes back towards p_0 from p_3 (assume it's r_0). The scenario right when r_0 arrives at p_3 at time $t \geq 2$ is depicted in Figure 2.

Let T be the earliest time after reaching p_3 that r_0 can be within $T(\sqrt{2} - 1)$ of the origin. At that time, r_0 will be exactly $1 - T(\sqrt{2} - 1)$ away from p_3. Since it left p_3 at time at least 2, this means that it is also at most $T - 2$ away from p_3. Thus, we have that $1 - T(\sqrt{2} - 1) \leq T - 2$. Solving this for T gives that $T \geq \frac{3\sqrt{2}}{2}$.

At this time, r_1 must still be within $T(1 - \sqrt{2})$ of the origin, since until T it was the only robot available to satisfy our assumption that there is always some robot within $t(1 - \sqrt{2})$ of the origin.

After this time, however, now that r_0 is sufficiently close to the origin, r_1 can immediately walk at full speed to p_4. It will still take r_1 at least $1 - T(\sqrt{2} - 1)$ additional time to reach p_4, for a total time of:

$$T + 1 - T(\sqrt{2} - 1) = 1 + T(2 - \sqrt{2}) \geq 1 + 3(\sqrt{2} - 1) = 3\sqrt{2} - 2 =: R$$

Thus, as long as there is always a robot within $t(\sqrt{2} - 1)$ of the origin, it is not possible to satisfy both requests before time $R \coloneqq 3\sqrt{2} - 2$. ◀

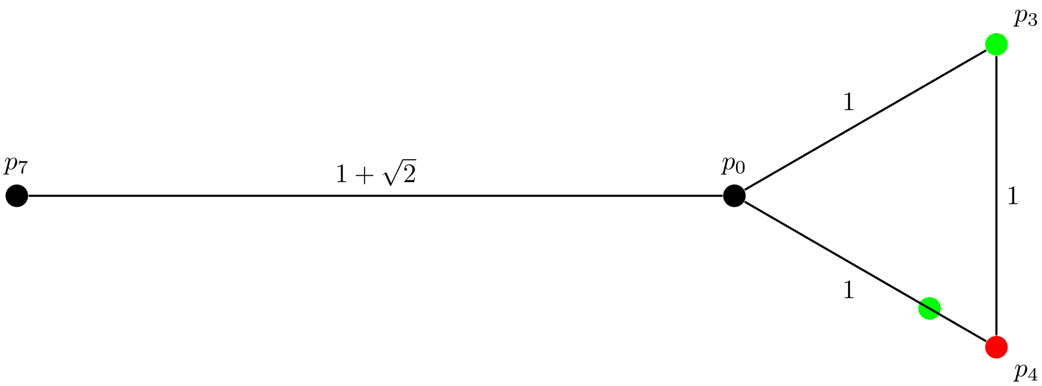


Figure 2 Robots at time $t = 2$, where r_1 is $2(\sqrt{2} - 1)$ away from p_0.

We'll need to verify some other simple but somewhat strange-looking facts.

▶ **Lemma 3.** *There exists a schedule for the input σ_A on M which unfreezes all robots by time $t = 1$, and another schedule where a single robot unfreezes every other robot by time $t = 2$. Moreover, M has $N - 1 := 1$ additional edges of length $1 + \sqrt{2}$ connected to p_0, on which none of the requests in σ_A occur.*

Proof. The two robots could unfreeze both by going to p_3 and p_4 right away. One robot could complete σ_A by first visiting p_3, and then p_4, taking 2 time units. Also, M has the edge connecting p_0 and p_7 of length $1 + \sqrt{2}$ that is not used by σ_A. ◀

Now, we can consider the two cases given by Lemma 2 to prove a lower bound.

▶ **Lemma 4.** *Let $R \leq 1 + \sqrt{2}$ be a real number, $N \geq 2$ an integer. Suppose M is a metric such that for any algorithm A, there exists an input σ_A with N robots at p_0 at time $t = 0$, such that Lemma 2 and Lemma 3 both hold. Then the competitive ratio of any online algorithm on M is at most R.*

Proof. Take any algorithm A on M and let σ_A be the input given by the hypotheses. By Lemma 2, we know that either A takes at least R time on σ_A, or there exists some time $1 + \sqrt{2} \geq t \geq 2$ when only $N - 2$ robots are within $t(\sqrt{2} - 1)$ of the origin. We will consider each of these as separate cases.

Case 1: A takes at least R time to complete σ_A. For σ_A, we know from Lemma 3 that the optimal scheduling tree finishes by time 1, so $OPT(\sigma_A) \leq 1$. This gives a competitive ratio of $\frac{A(\sigma_A)}{OPT(\sigma_A)} \geq R$.

Case 2: There exists some time $2 \leq t \leq 1 + \sqrt{2}$ when only $N - 2$ robots are closer than $t(\sqrt{2} - 1)$ to p_0. Let $p_1, \ldots, p_{n-1}$ be the endpoints other than p_0 of the edges of M given Lemma 3. Now, we modify σ_A to add the additional requests $(p_i \cdot t(\sqrt{2} - 1), t)$, which occur at time t along the edge from p_0 to p_i located at a distance of t away from p_0, for $i = 1, 2, \ldots, n - 1$. Then the optimal schedule can complete in time t. It starts by sending one robot to complete σ_A by time 2 using Lemma 3, and the other $N - 1$ robots to complete the extra requests at time $t \geq 2$.

For A, when the last request is released at time t, there are only $N - 2$ robots closer than $t(\sqrt{2} - 1)$ to p_0. If there are any robots on the edges connecting p_0 and p_i for $i = 1, \ldots, n - 1$, then they are no closer than p_0 is to a frozen robot, so by Lemma 2, they are counted among the $N - 2$. Now, $N - 2$ robots cannot complete the additional requests in time less than $2t$, which is far too slow. Therefore some robot not among the $N-2$ must unfreeze one of the new robots. Combining the hypotheses of Lemmas 2 and 3, the shortest route this robot can take goes through p_0. This robot (and therefore A) must spend a total time of at least $t + t(\sqrt{2}-1) + t = t(1+\sqrt{2})$ total time to satisfy that request, giving a competitive ratio of at least $\frac{t(1+\sqrt{2})}{t} = 1+\sqrt{2}$.

Since $R \le 1+\sqrt{2}$, the competitive ratio for any algorithm A is not less than R. ◀

If we apply Lemma 4 to our particular metric M and σ_A, it proves that no competitive ratio better than $R := 3\sqrt{2}-2$ is possible. As said before, we could optimize the analysis in this case to show a better bound, but since we will prove a tight bound in the next section, we won't bother. Since we have phrased Lemma 4 in such a general manner, it suffices to construct metrics M where Lemma 3 holds and Lemma 2 can be proven for a smaller values of R.

5 Tight Lower Bound

Let k be a non-negative integer. We will define a metric M_k with parameters N_k, a natural number, and T_k, a rooted tree on N_k vertices that has depth k, both of which are a function of k.

Let's construct a weighted graph which will form our metric M_k. Let $p_1, p_2, \ldots, p_{N_k-1}$ be vertices of degree one connected to a vertex p_0 by edges of length $1+\sqrt{2}$. Then make N_k+1 copies of the tree T_k (to be described), rooted at vertices $p_{N_k}, p_{N_k+1}, \ldots, p_{2N_k}$. Finally, connect all vertices in these trees to p_0 by edges of length 1.

We will describe the tree T_k recursively. We will define N_k to always be the number of vertices in the tree T_k, and so also achieve a recursive description of N_k. The tree T_0 is a single point, and T_1 consists of two vertices connected by an edge of length one. Then for $k > 1$, let T_{k+1} be rooted at a vertex v_0, with descendants $v_1, \ldots, v_{N_k}$. All of the edges connecting v_0 and v_i have length 1/2 For all i, let v_i be the root of a copy of T_k with all edge lengths halved. This completes the description of T_k and N_k, and so also M_k. A sketch of the tree T_k can be seen in Figure 3. Observe that $N_{k+1} = N_k^2 + 1$, that T_k has $k+1$ layers $0, 1, \ldots, k$ (where layer 0 is just the root), but that any path from the root to a leaf has length exactly 1, for $k \ge 1$. Also, note that M_1 is exactly the metric used in the Section 4.

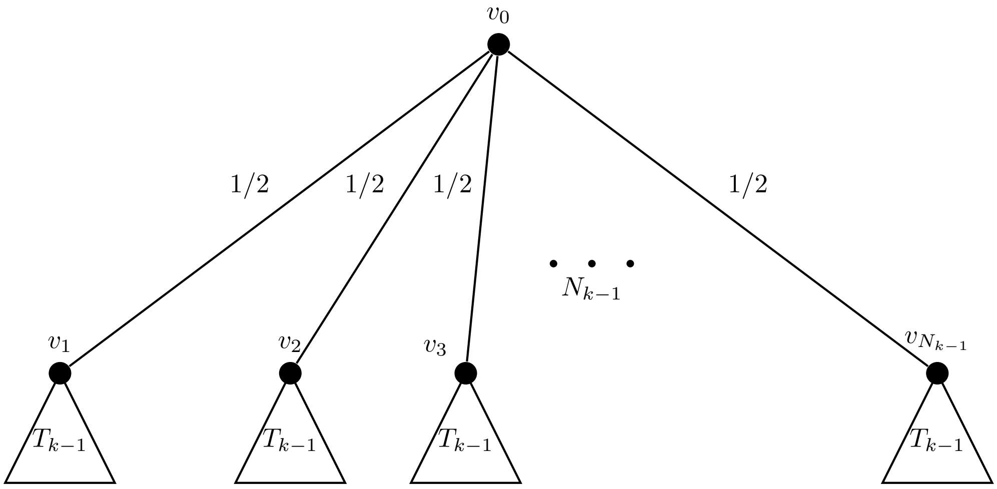


Figure 3 The tree T_k, in terms of T_{k-1}.

Let's define the input σ_A, for A an arbitrary online algorithm for the metric M_k. Suppose there are N_k starting robots active at p_0 at time 0, and no frozen robots. Observe the positions of the robots at time $t = 1$. Then since there are $N_k + 1$ copies of T_k, at least one tree will have no robots along any of the edges to any of the vertices in the tree. Without loss of generality assume this is the tree rooted at p_{N_k}. Then let σ_A be the input which starts N_k robots active at p_0 at time 0, and releases robots at time $t = 1$ at each node of the tree rooted at p_{N_k}. Namely, if a node of T_k has down-degree d, then release $\max(d - 1, 1)$ frozen robots at that node. The only exception will be the root of T_k, which has down-degree N_{k-1}, but σ_A releases N_{k-1} frozen robots at the root instead of $N_{k-1} - 1$ (this extra robot will make the analysis slightly more clean). The idea behind this construction of σ_A is that it provides just enough robots such that if the root were unfrozen, the robots could cascade through the tree unfreezing everything in 1 unit time.

▶ **Lemma 5.** *The number of frozen robots released by σ_A in layers $0, \ldots, i - 1$ of the tree is equal to the number of nodes in layer i.*

Proof. We use induction on i. For the base case $i = 1$, the number of frozen robots in layer 0 is just the number of frozen robots at the root, which by construction is N_k, which is also the number of nodes in layer 1 of T_k.

Suppose the lemma holds for some $i \geq 1$. In layer i, each node has down degree d and $d - 1$ robots located at it (since $i \geq 1$). Thus, the total number of nodes in layer i layer plus the number of robots in layer i is equal to the number of nodes in layer $i + 1$. By our inductive hypothesis, then, the total number of robots in layers $0, \ldots, i$ is equal to the number of nodes in layer $i + 1$. ◀

We aim to prove versions of Lemmas 2 and 3 for M_k, for some real number $R_k \in [2, 1 + \sqrt{2}]$ depending on k. The previous section provides the base case $k = 1$, where $R_1 = 3\sqrt{2} - 2$. Also note that everything holds for $k = 0$, when the graph consists of three spokes, and $R_0 = 2$. In general, we will define $R_k := 1 + \sqrt{2} - (\sqrt{2} - 1)^{k+1}$. One can check that this matches R_0 and R_1. This formula has the important properties that $R_{k+1} - R_k < 2^{-(k+1)}$ and that $\lim_{k\to\infty} R_k = 1 + \sqrt{2}$, which are both clear. This definition isn't just arbitrary though; it arises as the amount of time it takes to bounce back and forth between T_k and the ball of radius $t(\sqrt{2} - 1)$ around p_0.

▶ **Lemma 6.** *Suppose that at all times $t \in [R_k, R_{k+1})$ there are at least $N_k - 1$ robots within a distance $t(\sqrt{2} - 1)$ of p_0. Then there exist $N_k - 1$ robots that are unfrozen at time R_k which do not unfreeze any robots at any time $t \in [R_k, R_{k+1}]$.*

Proof. A similar argument to what follows appeared within the proof of Lemma 2.

At time R_k, there exist at least $N_k - 1$ robots that are within $R_k(\sqrt{2} - 1)$ of p_0. If these robots stay within $t(\sqrt{2} - 1)$ of p_0, then they will never reach a frozen robot in this time interval, since all frozen robots are at least one away from p_0 and $R_{k+1} < 1 + \sqrt{2} = 1/(\sqrt{2} - 1)$. Then at some time t some robot that either was frozen at time R_k or unfroze another robot must enter the ball of radius $t(\sqrt{2} - 1)$, otherwise there will always be these $N_k - 1$ robots that never unfreeze other robots. Now, since this robot entering the ball must have come from a point at a distance 1 from p_0, the minimal time $x > 0$ after R_k required such that $1 - x \leq (x + R_k)(\sqrt{2} - 1)$ is $x = \frac{1 - R_k(\sqrt{2} - 1)}{\sqrt{2}}$. Then, a robot exiting the ball has a distance x remaining to reach a frozen robots, which therefore cannot occur until time at least $R_k + 2x$.

Now, it can be checked that in fact, $R_{k+1} = R_k + \sqrt{2}(1 - R_k(\sqrt{2} - 1)) = R_k + 2x$. Therefore it is impossible to avoid having $N_k - 1$ robots never unfreeze a robot in this time interval. ◀

▶ **Lemma 7.** *Lemma 3 holds for the metric M_k with input σ_A and the integer N_k.*

Proof. A schedule can unfreeze all the robots released in σ_A by time $t = 1$ by having each of our starting N_k robots go directly from p_0 to a different vertex of the tree rooted at p_{N_k}. At time 1, they will all arrive and wake up all of the robots.

We can also have a single robot unfreeze all of the robots by time $t = 2$. First, our one robot moves to p_{N_k} by time $t = 1$. Then, each of the newly unfrozen robots moves down the tree to the root of some copy of T_{k-1}. There were N_k robots frozen at p_{N_k}, so all of the N_k roots can be reached by time $t = 3/2$. By induction, T_{k-1} can be traversed by a single robot in one time unit (the base case $k = 1$ is clear), but these copies of T_{k-1} have edges of half the length, so all of these robots can be traversed in half a time unit. Therefore the entire tree can be visited by time $t = 2$.

Finally, the edge from p_0 to the vertices $p_1, p_2, \ldots, p_{N_k-1}$ give us $N_k - 1$ additional edges of length $1 + \sqrt{2}$ on which no requests from σ_A occur. ◀

It now remains to prove a sufficiently strong version of Lemma 2, so that we can use Lemma 4. Call an unfrozen robot *free* at time t if it is more than $t(\sqrt{2} - 1)$ away from p_0. In the context of Lemma 2, we are looking for a time t when all but possibly $N_k - 2$ robots are free. It will be helpful to use this to constrain how many robots can be free by a certain time.

▶ **Lemma 8.** *Suppose A is an algorithm such that at any time under the input σ_A, there are at least $N_k - 1$ robots which are not free. Then A cannot unfreeze as many robots as there are nodes in layers $0, \ldots, i$ before time R_i.*

Proof. We use induction on i. The base case, $i = 0$, simply says that A cannot unfreeze any robots before time 2. This is true because σ_A chooses to put all of the frozen robots on a tree that no active robot is near. The nearest a robot could be at $t = 1$ is p_0, which is distance 1 away from any node of the tree.

Let x_i be the number of nodes in layers $0, \ldots, i$. Suppose that our lemma holds for $i - 1$. Then just before time R_{i-1}, there are at most x_{i-1} free robots, since of our initial N_k robots at least $N_k - 1$ of them must be near p_0 and not be free.

By Lemma 6, there must be $N_k - 1$ robots which were unfrozen before R_{i-1} that never unfreeze any robots between the times R_{i-1} and R_i. In particular, without loss of generality we will assume that these are the $N_k - 1$ robots that are required to stay near p_0, so the only robots we have available during this interval are the x_{i-1} free robots.

Since $R_i - R_{i-1} < 2^{-i}$, and every edge coming out of a node in one of layers $0, \ldots, i - 1$ has length at least 2^{-i}, it is not possible for any free robot to fully traverse one of these edges between times R_i and R_{i-1}. Thus, we can assume these edges don't exist for bounding the number of robots that can be woken up between times R_i and R_{i-1}. Ignoring these edges, we have a bunch of subtrees in the bottom layers, each of which have a total of N_{k-i} robots on them, and x_{i-1} single nodes in layers $0, \ldots, i - 1$ which each have at least N_{k-i} robots frozen on them.

Each of our x_{i-1} robots can wake up at most all of the robots in either one subtree or one single node. Since every node in layers $0, \ldots, i - 1$ has at least N_{k-i} robots at it, and each of the subtrees has at most N_{k-i} robots in it, the number of robots that can be woken up is maximized by sending a free robot to each of the nodes in layers $0, \ldots, i - 1$. By Lemma 5, the total number of robots in these layers equal to the number of nodes in layer i. Thus, we now have a total of x_i free robots, so we cannot have more than x_i free robots before time R_i.

If we have x_i free robots, then since we started with 1 free robot, we have unfrozen less than x_i robots by this point. ◀

▶ **Corollary 1.** *Lemma 2 holds on the metric M_k with input σ_A, the integer N_k, and $R_k := 1 + \sqrt{2} - (\sqrt{2} - 1)^{k+1}$.*

Proof. Suppose that at every time $t \in [2, 1 + \sqrt{2}]$ there exists at least $N_k - 1$ robots at a distance more than $t(\sqrt{2} - 1)$ from p_0, otherwise the result is immediate. Then we can apply Lemma 8 with $i = k$ to get that the number of robots unfrozen must not be more than there are nodes in T_k (which is N_k) before time R_k. Now, the number of frozen robots released in σ_A is more than the number of nodes in T_k, since each node has at least one frozen robot but most have more. Therefore $A(\sigma) \geq R_k$. ◀

Proof of Theorem 2.2.
First, Theorem 3.1 says that our algorithm is $(1 + \sqrt{2})$-competitive.

Due to Corollary 1 and Lemma 7, we can use Lemma 4 on M_k, with input σ_A, integer N_k, and $R_k := 1 + \sqrt{2} - (\sqrt{2} - 1)^{k+1}$. Then any online algorithm on M_k achieves a competitive ratio of at most R_k. Fix $\varepsilon > 0$. Since $\lim_{k \to \infty} R_k = 1 + \sqrt{2}$, choose k such that $1 + \sqrt{2} - R_k < \varepsilon$. Therefore no algorithm is $(1 + \sqrt{2} - \varepsilon)$-competitive on M_k. ◀

Our analysis required trees which have size exponential in $1/\varepsilon$, since N_k is doubly exponential in k and $1 + \sqrt{2} - R_k$ is singly exponential in k. Could there be an algorithm that is on the order of $(1 + \sqrt{2} - O(\frac{1}{\log n}))$-competitive, for metrics coming from weighted graphs on at most n vertices?

References

1 Esther M Arkin, Michael A Bender, Sándor P Fekete, Joseph SB Mitchell, and Martin Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46(2):193–221, 2006.

2 Esther M Arkin, Michael A Bender, and Dongdong Ge. Improved approximation algorithms for the freeze-tag problem. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 295–303. ACM, 2003.

3 Mikael Hammar, Bengt J Nilsson, and Mia Persson. The online freeze-tag problem. In *Latin American Symposium on Theoretical Informatics*, pages 569–579. Springer, 2006.

4 Jochen Könemann, Asaf Levin, and Amitabh Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41(2):117–129, 2005.

5 Zahra Moezkarimi and Alireza Bagheri. A PTAS for geometric 2-FTP. *Information Processing Letters*, 114(12):670–675, 2014.

6 Marcelo O Sztainberg, Esther M Arkin, Michael A Bender, and Joseph SB Mitchell. Analysis of heuristics for the freeze-tag problem. In *Scandinavian Workshop on Algorithm Theory*, pages 270–279. Springer, 2002.

7 Ehsan Najafi Yazdi, Alireza Bagheri, Zahra Moezkarimi, and Hamidreza Keshavarz. An O(1)-approximation algorithm for the 2-dimensional geometric freeze-tag problem. *Information Processing Letters*, 115(6-8):618–622, 2015.

Magic: The Gathering Is Turing Complete

Alex Churchill
Independent Researcher, UK
http://www.myhomepage.edu
alex.churchill@cantab.net

Stella Biderman
LucyLabys, Georgia Institute of Technology, Atlanta, GA, USA
Booz Allen Hamilton, Atlanta, USA
http://www.stellabiderman.com
stellabiderman@gatech.edu

Austin Herrick
Penn Wharton Budget Model, University of Pennsylvania, Philadelphia, PA, USA
aherrick@wharton.upenn.edu

Abstract

Magic: The Gathering is a popular and famously complicated trading card game about magical combat. In this paper we show that optimal play in real-world *Magic* is at least as hard as the Halting Problem. This provides a positive answer to the question "is there a real-world game where perfect play is undecidable under the rules in which it is typically played?", a question that has been open for a decade [1, 9]. To do this, we present a methodology for embedding an arbitrary Turing machine into a game of *Magic* such that the first player is guaranteed to win the game if and only if the Turing machine halts. Our result applies to how real *Magic* is played, can be achieved using standard-size tournament-legal decks, and does not rely on stochasticity or hidden information. Our result is also highly unusual in that all moves of both players are forced in the construction. This shows that even recognising who will win a game in which neither player has a non-trivial decision to make for the rest of the game is undecidable. We conclude with a discussion of the implications for a unified computational theory of games and remarks about the playability of such a board in a tournament setting.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Turing machines, computability theory, Magic: the Gathering, two-player games

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.9

Related Version A preprint of this paper is available at https://arxiv.org/abs/1904.09828.

Acknowledgements We are grateful to C-Y. Howe for help simplifying our Turing machine construction considerably and to Adam Yedidia and Edwin Thomson for conversations about the design and construction of Turing machines.

1 Introduction

Magic: The Gathering (also known as *Magic*) is a popular trading card game owned by Wizards of the Coast. Formally, it is a two-player zero-sum stochastic card game with imperfect information, putting it in the same category as games like poker and hearts. Unlike those games, players design their own custom decks out of a card-pool of over 20,000 unique cards. *Magic*'s multifaceted strategy has made it a popular topic in artificial intelligence research.

In this paper, we examine *Magic: The Gathering* from the point of view of algorithmic game theory, looking at the computational complexity of evaluating who will win a game. As most games have finite limits on their complexity (such as the size of a game board)

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 9; pp. 9:1–9:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

most research in algorithmic game theory of real-world games has primarily looked at generalisations of commonly played games rather than the real-world versions of the games. A few real-world games have been found to have non-trivial complexity, including *Dots-and-Boxes*, *Jenga* and *Tetris* [7]. We believe that no real-world game is known to be harder than NP previous to this work.

Even when looking at generalised games, very few examples of undecidable games are known. On an abstract level, the Team Computation Game [8] shows that some games can be undecidable, if they are a particular kind of team game with imperfect information. The authors also present an equivalent construction in their Constraint Logic framework that was used by Coulombe and Lynch (2018) [6] to show that some video games, including *Super Smash Bros Melee* and *Mario Kart*, have undecidable generalisations. Constraint Logic is a highly successful and highly flexible framework for modelling games as computations.

The purpose of this paper is to present the first proof that a game, *as it is actually played in the world*, can be undecidable. The core of this paper is the construction presented in Section 4: a universal Turing machine embedded into a game of *Magic: The Gathering*. As we can arrange for the victor of the game to be determined by the halting behaviour of the Turing machine, this construction establishes the following theorem:

▶ **Theorem 1.** *Determining the outcome of a game of Magic: The Gathering in which all remaining moves are forced is undecidable.*

1.1 Previous Work

Prior to this work, no undecidable real games were known to exist. Demaine and Hearn (2009) [9] note that almost every real-world game is trivially decidable, as they produce game trees with only computable paths. They further note that Rengo Kriegspiel[1] is "a game humans play that is not obviously decidable; we are not aware of any other such game." It is conjectured by Auger and Teytaud (2012) [1] that Rengo Kriegspiel is in fact undecidable, and it is posed as an open problem to demonstrate any real game that is undecidable.

The approach of embedding a Turing machine inside a game directly is generally not considered to be feasible for real-world games [9]. Although some open-world sandbox digital games such as Minecraft and Dwarf Fortress can support the construction of Turing machines, those machines have no strategic relevance and those games are deliberately designed to support large-scale simulation. In contrast, leading formal theory of strategic games claims that the unbounded memory required to simulate a Turing machine entirely in a game would be a violation of the very nature of a game [8].

The computational complexity of *Magic: The Gathering* has been studied previously by several authors. Our work is inspired by [4], in which it was shown that four-player *Magic* can simulate a Turing machine under certain assumptions about player behaviour. In that work, Churchill conjectures that these limitations can be removed and preliminary work along those lines is discussed in [5]. The computational complexity of checking the legality of a particular decision in *Magic* (blocking) is investigated in [3] and is found to be coNP-complete. There have also been a number of papers investigating algorithmic and artificial intelligence approaches to playing *Magic*, including Ward and Cowling (2009) [14], Cowling et al. (2012) [15], and Esche (2018) [10]. Esche (2018) briefly considers the theoretical computational complexity of *Magic* and states an open problem that has a positive answer only if *Magic* end-games are decidable.

[1] Rengo Kriegspiel is a combination of two variations on Go: Rengo, in which two players play on a team alternating turns, and Shadow Go, in which players are only able to see their own moves.

1.2 Our Contribution

This paper completes the project started by Churchill (2012) [4] and continued by Churchill et al. (2014) [5] of embedding a universal Turing machine in *Magic: The Gathering* such that determining the outcome of the game is equivalent to determining the halting of the Turing machine. This is the first result showing that there exists a real-world game for which determining the winning strategy is non-computable, answering an open question of Demaine and Hearn [9] and Auger and Teytaud [1] in the positive.

Our result is also a contribution to the existing literature on the computational complexity of *Magic: the Gathering* [4, 3, 10]. Combined with Rice's Theorem [12], it also answers an open problem from Esche [10] in the negative by showing that the equivalence of two strategies for playing *Magic* is undecidable.

This result raises important foundational questions about the nature of a game itself. As we have already discussed, the leading formal theory of games holds that this construction is unreasonable, if not impossible, and so a reconsideration of those assumptions is called for. In section 5.1 we discuss additional foundational assumptions of Constraint Logic that *Magic: The Gathering* violates, and present our interpretation of the implications for a unified theory of games.

1.3 Overview

The paper is structured as follows. In Section 2 we provide background information on this work, including previous work on *Magic* Turing machines. In Section 3 we present a sketch of the construction and its key pieces. In Section 4 we provide the full construction of a universal Turing machine embedded in a two-player game of *Magic*. In Section 5 we discuss the game-theoretic and real-world implications of our result.

Appendix A provides a brief overview of the relevant rules to *Magic* for those who are not familiar with the game.

2 Preliminaries

One initial challenge with *Magic: The Gathering* is the encoding of information. Some cards ask players to choose a number. Although rules for how to specify a number are not discussed in the *Comprehensive Rules* [16], convention is that players are allowed to specify numbers in any way that both players can agree to. For example, you are allowed to choose the number 2^{100} or $\lceil \log 177 \rceil$. This presents an issue brought to our attention by Fortanely [11]. Consider the following situation: both players control **Lich**, **Transcendence**, and **Laboratory Maniac**. One player then casts **Menacing Ogre**. The net effect of this play is the "Who Can Name the Bigger Number" game – whoever picks the biggest number wins on the spot. If players are allowed to use sufficiently complicated (but well-defined) functions to express their choices, identifying the next board state can be non-computable even given the players' choices [2]. To remedy this situation, we require that any numbers specified by a player must be expressed in standard binary notation[2].

We believe that with this restriction *Magic: The Gathering* is *transition-computable*, meaning that the function that maps a board state and a move to the next board state is computable[3]. However, it is unclear how to prove this beyond exhaustive analysis of the over 20,000 unique cards in the game. We leave that question open for future work:

[2] Only integers exist in *Magic*. If a result would be a non-integer, it is rounded.

[3] We avoid the term "computable game" which is more commonly used to mean that the game has a computable winning strategy.

▶ Conjecture 1. The function that takes a board state and a legal move and returns the next board state in *Magic: The Gathering* is computable.

In this conjecture we say "a legal move" because it is also not obvious that checking to see if a move is legal is computable. Chatterjee and Ibsen-Jensen [3] show that checking the legality of a particular kind of game action in *Magic: the Gathering* is coNP-complete, but the question has not been otherwise considered. Again, we leave this for future work:

▶ Conjecture 2. There does not exist an algorithm for checking the legality of a move in *Magic: The Gathering.*

2.1 Previous *Magic* Turing Machines

In Churchill (2012) [4], the author presents a *Magic: The Gathering* end-game that embeds a universal Turing machine. However, this work has a major issue from a computability theory point of view: it's not quite deterministic. At several points in the simulation, players have the ability to stop the computation at any time by opting to decline to use effects that say "may." For example, **Kazuul Warlord** reads "Whenever Kazuul Warlord or another Ally enters the battlefield under your control, you may put a $+1/+1$ counter on each Ally you control." Declining to use this ability will interfere with the Turing machine, either causing it to stop or causing it to perform a different calculation from the one intended. The construction as given in Churchill (2012)[4] works under the assumption that all players that are given the option to do something actually do it, but as the author notes it fails without this assumption. Attempts to correct this issue are discussed in Churchill et al. [5].

In this work, we solve this problem by reformulating the construction to exclusively use cards with mandatory effects. We also substantially simplify the most complicated aspect of the construction, the recording of the tape, and reduce the construction from one involving four players to one involving two, and which only places constraints on one player's deck, matching the format in which *Magic* is most commonly played in the real world (two-player duels). Like the previous work, we will embed Rogozhin's (2, 18) universal Turing machine [13].

3 An Overview of the Construction

In this section we give a big picture view of the Turing machine, with full details deferred to the next section[4]. The two players in the game are named Alice and Bob.

To construct a Turing machine in *Magic: The Gathering* requires three main elements: the tape which encodes the computation, the controller which determines what action to take next based on the current state and the last read cell, and the read/write head which interacts with the tape under the control of the controller.

3.1 The Tape

As the rules of *Magic: The Gathering* do not contain any concept of geometry or adjacency, encoding the tape itself is tricky. Our solution is to have many creature tokens with carefully controlled power and toughness, with each token's power and toughness representing the distance from the head of the Turing machine. The tape to the left of the Turing machine's current read head position is represented by a series of creature tokens which all have

[4] A walkthrough of the construction

the game colour green, while the tape to the right is represented by white tokens. Our distance-counting starts at 2, so there is one 2/2 token representing the space currently under the head of the Turing machine; a green 3/3 token represents the tape space immediately to the left of the Turing head, a green 4/4 is the space to the left of that, and so on. Rogozhin's universal Turing machine starts with the read head in the middle of the tape [13].

To represent the symbols on the tape, we use creature types. We choose 18 creature types from the list of creature types in *Magic* to correspond to the 18 symbols in Rogozhin's (2, 18) UTM. We can choose these creature types to begin with successive letters of the alphabet: Aetherborn, Basilisk, Cephalid, Demon, Elf, Faerie, Giant, Harpy, Illusion, Juggernaut, Kavu, Leviathan, Myr, Noggle, Orc, Pegasus, Rhino, and Sliver. For example, a green 5/5 Aetherborn token represents that the 1^{st} symbol is written on the 3^{rd} cell to the left of the head, and a white 10/10 Sliver represents that the 18^{th} symbol is written on the 8^{th} cell to the right of the head. These tokens are all controlled by Bob, except the most recently created token (the space the Turing head has just left) which is controlled by Alice.

3.2 The Controller

Control instructions in a Turing machine are represented by a table of conditional statements of the form "if the machine is in state s, and the last read cell is symbol k, then do such-and-such." Many *Magic* cards have triggered abilities which can function like conditional statements. The two we shall use are **Rotlung Reanimator** ("Whenever Rotlung Reanimator or another Cleric dies, create a 2/2 black Zombie creature token") and **Xathrid Necromancer** ("Whenever Xathrid Necromancer or another Human creature you control dies, create a tapped 2/2 black Zombie creature token"). We will use both, and the difference between tapped and untapped creature tokens will contribute to the design of the Turing machine[5].

Each **Rotlung Reanimator**[6] needs to trigger from a different state being read – that is, a different creature type dying – and needs to encode a different result. Fortunately, *Magic* includes cards that can be used to edit the text of other cards. The card **Artificial Evolution** is uniquely powerful for our purposes, as it reads "Change the text of target spell or permanent by replacing all instances of one creature type with another. The new creature type can't be Wall. *(This effect lasts indefinitely.)*" So we create a large number of copies of **Rotlung Reanimator** and edit each one. A similar card, **Glamerdye**, can be used to modify the colour words within card text.

Thus, we edit a **Rotlung Reanimator** by casting two copies of **Artificial Evolution** replacing "Cleric" with "Aetherborn" and "Zombie" with "Sliver" and one copy of **Glamerdye** to replace "black" with "white", so that this **Rotlung Reanimator** now reads "Whenever Rotlung Reanimator or another *Aetherborn* dies, create a 2/2 *white Sliver* creature token"[7]. This **Rotlung Reanimator** now encodes the first rule of the q_1 program of the (2, 18) UTM: "When reading symbol 1 in state A, write symbol 18 and move left." The Aetherborn creature token represents symbol 1, the Sliver creature token represents symbol 18, and the fact that the token is white leads to processing that will cause the head to move left.

We similarly have seventeen more **Rotlung Reanimator**s encoding the rest of the q_1 program from [13]. Between them they say:

[5] See Appendix A.3 for information about tapping

[6] For now we will speak about **Rotlung Reanimator** for simplicity. Some of these will in fact be **Xathrid Necromancer**s as explained in the next section.

[7] Throughout this paper, card text that has been modified is written in italics.

1. Whenever an *Aetherborn* dies, create a 2/2 *white Sliver*.

2. Whenever a *Basilisk* dies, create a 2/2 *green Elf*.

Whenever a ... dies, create a 2/2 ...

18. Whenever a *Sliver* dies, create a 2/2 *green Cephalid*.

See Table 2 in Appendix B for the full encoding of the program.

3.3 The Read/Write Head

The operation "read the current cell of the tape" is represented in-game by forcing Alice to cast **Infest** ("All creatures get –2/–2 until end of turn."), which causes the unique token with 2 toughness to die. It had a colour (green or white) which is irrelevant, and a creature type which corresponds to the symbol written on that cell. That creature type is noticed by a **Rotlung Reanimator**, which has a triggered ability that is used to carry out the logic encoded in the head of the Turing machine. It produces a new 2/2 token, containing the information written to the cell that was just read.

The Turing machine then moves either left or right and modifies the tokens to keep the tape in order by adding $+1/+1$ counters to all tokens on one side of the head and –1/–1 counters to all tokens on the other side. Moving left or right will be accomplished by casting first **Cleansing Beam** ("Radiance – Cleansing Beam deals 2 damage to target creature and each other creature that shares a colour with it.") and then **Soul Snuffers** ("When Soul Snuffers enters the battlefield, put a -1/-1 counter on each creature").

3.4 Adding a Second State

Everything described so far outlines the operation of one state of the Turing machine. However, our Turing machine requires two states. To accomplish this, we leverage *phasing*. An object with phasing can "phase in" or "phase out", and while it's phased out, it's treated as though it doesn't exist. We can grant phasing to our **Rotlung Reanimator**s using the enchantment **Cloak of Invisibility** ("Enchanted creature has phasing and can't be blocked except by Walls") and create a second set of **Rotlung Reanimator**s to encode the program q_2. This second set will also be granted phasing in this way, and the two programs will be set to be in opposition, with one phased out and one phased in at every point in time. At the moment we read the current cell, the set of **Rotlung Reanimator**s that is phased in is determined by which state we are in.

Objects with phasing phase in or out at the beginning of their controller's turn, effectively toggling between two states. Accordingly we will arrange for the turn cycle to last 4 turns for each player when no state change occurs, but just 3 turns when we need to change state.

4 The Full Construction

Now we will provide the full construction of the *Magic: The Gathering* Turing machine and walk through a computational step. The outline of one step of the computation is as follows (Bob's turns are omitted as nothing happens during them):

T1 Alice casts **Infest**. Turing processing occurs: a white or green token dies, a new white or green token is created.

T2 Alice casts **Cleansing Beam**, putting two $+1/+1$ counters on the side of the tape we are moving away from.

T3 If the Turing machine is remaining in the same state, Alice casts **Coalition Victory**. If it is changing state, Alice casts **Soul Snuffers**, putting a –1/–1 counter on each creature.

T4 If the Turing machine is remaining in the same state, this is the point where Alice casts **Soul Snuffers**. Otherwise, the next computational step begins.

After all four turns have passed, the Turing machine has finished processing for one step of the computation and moves on to the next.

4.1 Beginning a Computational Step and Casting Spells

At the beginning of a computational step, it is Alice's turn and she has the card **Infest** in hand. Her library consists of the other cards she will cast during the computation (**Cleansing Beam**, **Coalition Victory**, and **Soul Snuffers**, in that order). Bob's hand and library are both empty. The Turing machine is in its starting state and the tape has already been initialised.

At the start of each of Alice's turns, she has one card in hand. She's forced to cast it due to Bob controlling **Wild Evocation**, which reads "At the beginning of each player's upkeep, that player reveals a card at random from their hand. If it's a land card, the player puts it onto the battlefield. Otherwise, the player casts it without paying its mana cost if able." When the card resolves, it would normally be put into her graveyard, but Alice is enchanted by **Wheel of Sun and Moon** ("Enchant player. If a card would be put into enchanted player's graveyard from anywhere, instead that card is revealed and put on the bottom of that player's library."), which causes it to be placed at the bottom of her library instead, allowing her to redraw it in the future and keeping the cards she needs to cast in order. After her upkeep step, Alice proceeds to her draw step and draws the card that she will cast next turn.

Alice has no choices throughout this process: she does control one land, but it remains permanently tapped because of **Choke** ("Islands don't untap during their controllers' untap steps"), so she is unable to cast any of the spells she draws except via **Wild Evocation**'s ability. Neither player is able to attack because they both control a **Blazing Archon** ("Creatures can't attack you.").

Bob has no cards in hand and controls **Recycle**, which reads (in part) "Skip your draw step". This prevents Bob from losing due to drawing from an empty library.

4.2 Reading the Current Cell

On the first turn of the cycle, Alice is forced to cast **Infest** ("All creatures get –2/–2 until end of turn."). This kills one creature: the tape token at the position of the current read head, controlled by Bob. This will cause precisely one creature of Bob's to trigger – either a **Rotlung Reanimator** or a **Xathrid Necromancer**. Which precise one triggers is based on that token's creature type and the machine's current state, corresponding to the appropriate rule in the definition of the (2, 18) UTM. This Reanimator or Necromancer will create a new 2/2 token to replace the one that died. The new token's creature type represents the symbol to be written to the current cell, and the new token's colour indicates the direction for the machine to move: white for left or green for right.

Alice controls **Illusory Gains**, an Aura which reads "You control enchanted creature. Whenever a creature enters the battlefield under an opponent's control, attach Illusory Gains to that creature." Each time one of Bob's **Rotlung Reanimator**s or **Xathrid Necromancer**s creates a new token, **Illusory Gains** triggers, granting Alice control of the newest token on the tape, and reverting control of the previous token to Bob. So at any point Bob controls all of the tape except for the most recently written symbol, which is controlled by Alice.

4.3 Moving Left or Right

If the new token is white, the Turing machine needs to move left. To do this we need to take two actions: put a $+1/+1$ counter on all white creatures (move the tape away from white), and put a –1/–1 counter on all green creatures (move the tape towards green). We rephrase this instead as: put two $+1/+1$ counters on all white creatures, and put a –1/–1 counter on *all* creatures.

On Alice's second turn, she casts **Cleansing Beam**, which reads "Cleansing Beam deals 2 damage to target creature and each other creature that shares a colour with it." Bob controls **Privileged Position** ("Other permanents you control have hexproof. (They can't be the targets of spells or abilities your opponents control.)") so none of Bob's creatures are a legal target. Alice controls some creatures other than the tape token, but they have all been granted creature type Assembly-Worker by a hacked **Olivia Voldaren**, and Alice controls a **Steely Resolve** naming Assembly-Worker ("Creatures of the chosen type have shroud. *(They can't be the targets of spells or abilities.)*") This makes it so that the only legal target for **Cleansing Beam** is the one tape token that Alice controls thanks to her **Illusory Gains**.

Recall that this token is white if we're moving left, or green if we're moving right. **Cleansing Beam** is about to deal 2 damage to each white creature if we're moving left, or to each green creature if we're moving right. Alice and Bob each control a copy of **Vigor** ("If damage would be dealt to another creature you control, prevent that damage. Put a $+1/+1$ counter on that creature for each 1 damage prevented this way."), so **Cleansing Beam** ends up putting two $+1/+1$ counters on either each white creature or each green creature.

On the last turn of the cycle, Alice casts **Soul Snuffers**, a 3/3 black creature which reads "When Soul Snuffers enters the battlefield, put a –1/–1 counter on each creature." There are two copies of **Dread of Night** hacked to each say "*Black* creatures get –1/–1", which mean that the **Soul Snuffers**' triggered ability will kill itself, as well as shrinking every other creature. The creatures comprising the tape have now received either a single –1/–1 counter, or two $+1/+1$ counters and a –1/–1 counter.

To ensure that the creatures providing the infrastructure (such as **Rotlung Reanimator**) aren't killed by the succession of –1/–1 counters each computational step, we arrange that they also have game colours green, white, red and black, using **Prismatic Lace**, "Target permanent becomes the colour or colours of your choice. *(This effect lasts indefinitely.)*" Accordingly, each cycle **Cleansing Beam** will put two $+1/+1$ counters on them, growing them faster than the –1/–1 counters shrink them. This applies to each creature except **Vigor** itself; to keep each player's **Vigor** from dwindling, there is a **Fungus Sliver** hacked to read "All *Incarnation* creatures have 'Whenever this creature is dealt damage, put a $+1/+1$ counter on it.' "

4.4 Changing State

The instruction to change state is handled by replacing seven of Bob's **Rotlung Reanimators** with **Xathrid Necromancer**. These two cards have very similar text, except that **Xathrid Necromancer** only notices Bob's creatures dying (this is not a problem, as the active cell of the tape is always controlled by Bob), and that **Xathrid Necromancer** creates its token *tapped.*

For example, when the q_1 program (State A) sees symbol 1, it writes symbol 18, moves left, and remains in state A. This is represented by a phasing **Rotlung Reanimator** under Bob's control saying "Whenever Rotlung Reanimator or another *Aetherborn* dies, create a 2/2 *white Sliver* creature token."

By contrast, when the q_1 program sees symbol 11, it writes symbol 12, moves left, and changes to state B. This is represented by a phasing **Xathrid Necromancer** under Bob's control saying "Whenever Xathrid Necromancer or another *Kavu* creature you control dies, create a tapped 2/2 *white Leviathan* creature token."

In both cases this token is created under Bob's control on turn T1, but Alice's **Illusory Gains** triggers and grants her control of it. In the case where it's tapped, that means at the beginning of turn T2, it will untap. This causes Alice's **Mesmeric Orb** ("Whenever a permanent becomes untapped, that permanent's controller puts the top card of his or her library into his or her graveyard.") to trigger, and that ability to be put on the stack at the same time as Bob's **Wild Evocation**'s trigger (since no player receives priority during the untap step). Alice is the active player, so Alice's trigger is put on the stack first and then Bob's[16]. This ensures that the **Wild Evocation** trigger resolves, forcing Alice to cast and resolve **Cleansing Beam**, before the **Mesmeric Orb** trigger resolves.

When the **Mesmeric Orb** trigger does resolve, it tries to put the **Coalition Victory** from the top of Alice's library into her graveyard. **Wheel of Sun and Moon** modifies this event to put **Coalition Victory** onto the bottom of her library, just underneath the **Cleansing Beam** that's just resolved.

Once all these triggers are resolved, Alice proceeds to her draw step. When the state is not changing, she will draw **Coalition Victory** at this point, but when the state is changing, that card is skipped and she moves on to draw **Soul Snuffers**.

The net result of this is that the computation step is 3 turns long for each player when the state is changing, but 4 turns long for each player when the state is not changing. In the normal 4-turn operation, Bob's phasing Reanimators and Necromancers will phase in twice and phase out twice, and be in the same state on one cycle's turn T1 as they were in the previous cycle's turn T1. But when changing state, they will have changed phase by the next cycle's turn T1, switching the Turing machine's state.

4.5 Out of Tape

The Turing tape can be initialised to any desired length before starting processing. But it is preferable to allow the machine to run on a simulated infinite tape: in other words, to assume that any uninitialised tape space contains symbol 3 (the blank symbol in the (2, 18) UTM), represented by creature type Cephalid. This is accomplished by having the ends of the currently-initialised tape marked by two special tokens, one green Lhurgoyf and one white Rat.

Suppose we've exhausted all the initialised tape to the left. This means that the casting of **Infest** on turn T1 kills the Lhurgoyf rather than one of the normal tape types. This does not directly trigger any of the normal Reanimators/Necromancers. Instead, Bob has another **Rotlung Reanimator** hacked to read "Whenever Rotlung Reanimator or another *Lhurgoyf* dies, create a 2/2 *green Lhurgoyf* creature token", and Alice has a **Rotlung Reanimator** hacked to read "Whenever Rotlung Reanimator or another *Lhurgoyf* dies, create a 2/2 *black Cephalid* creature token." Bob's trigger will resolve first, then Alice's.

First, Bob's Reanimator trigger creates a new Lhurgoyf just to the left of the current head. (Alice's **Illusory Gains** triggers and gives her control of this new Lhurgoyf, but that will soon change.) We have one copy of **Shared Triumph** set to Lhurgoyf ("Creatures of the chosen type get +1/+1") so this token arrives as a 3/3.

Second, Alice's Reanimator trigger now creates a 2/2 black Cephalid under Alice's control. The same two copies of **Dread of Night** as before are giving all black creatures –2/–2, so the black Cephalid will arrive as a 0/0 and immediately die.

The death of this Cephalid triggers one of the regular **Rotlung Reanimator**s of Bob's just as if a tape cell containing symbol 3 had been read: a new 2/2 token is created and **Illusory Gains** moves to that new token. The green Lhurgoyf token serving as an end-of-tape marker has been recreated one step over to the left.

The situation for the white Rat representing the right-hand end of the tape is exactly equivalent. Bob has a **Rotlung Reanimator** hacked to read "Whenever Rotlung Reanimator or another *Rat* dies, create a 2/2 *white Rat* creature token"; Alice has a **Rotlung Reanimator** hacked to read "Whenever Rotlung Reanimator or another *Rat* dies, create a 2/2 *black Cephalid* creature token"; and we have another **Shared Triumph** set to Rat. This algorithm would be a little more complex if reading symbol 3 could cause a state change in the (2, 18) UTM, but thankfully it cannot.

4.6 Halting

We choose to encode halting as making Alice win the game. When the Turing machine doesn't change state, Alice casts the card **Coalition Victory** on her third turn. It reads "You win the game if you control a land of each basic land type and a creature of each colour." This normally accomplishes nothing because she controls no blue creatures (**Prismatic Lace** has been used to give her creatures of all the other colours). She does, however, control one land, and also controls **Prismatic Omen**, which reads "Lands you control are every basic land type in addition to their other types."

When the halt symbol is read (symbol 17 in state A), the appropriate phasing Reanimator of Bob's reads "Whenever Rotlung Reanimator or another *Rhino* dies, create a 2/2 *blue Assassin* creature token." Alice's **Illusory Gains** takes control of this Assassin token in the usual way in turn T1. She now meets the condition for **Coalition Victory** when she casts it on turn T3, and wins the game. As the token is created by a **Rotlung Reanimator**, it will be untapped and so Alice will not be made to discard **Coalition Victory**.

If the encoded machine does not in fact halt then the game has entered an unbreakable deterministic infinite loop, which is specified as a draw by rule 104.4b [16].

5 Discussion

The construction in the previous section establishes Theorem 1: determining the outcome of a game of *Magic: The Gathering* in which all remaining moves are forced is undecidable. As determining the existence of a winning move is no harder than optimal play, this also establishes that optimal play in *Magic* is at least as hard as the halting problem.

The full complexity of optimal strategic play remains an open question, as do many other computational aspects of *Magic*. For example, a player appears to have infinitely many moves available to them from some board states of *Magic*. Whether or not there exists a real-world game of *Magic* in which a player has infinitely many *meaningfully different* moves available to them has the potential to impact the way we understand and model games as a form of computation. As far as we are aware, no existing computational models of games support games with infinitely many possible moves from a given board state.

5.1 Consequences for Computational Theories of Games

This construction establishes that *Magic: The Gathering* is the most computationally complex real-world game known in the literature, but it also raises several foundational questions about games as a form of computation. Some authors, such as Demaine and Hearn [8], have sought a formal framework for modelling games that is strictly sub-Turing. Unlike the open-world, non-strategic games in which Turing machines have been constructed before, *Magic: The Gathering* is unambiguously a two-player strategic game like such models attempt to represent. Therefore this result shows that any sub-Turing model is necessarily inadequate to capture all strategic games. Quite the opposite: it seems likely that a *super-Turing* model of games would be necessary to explain *Magic*. The naïve extensions of Demaine and Hearn's Constraint Logic to allow for unbounded memory appear to be meaningless, although it's possible that a clever approach would bring success.

▶ Open Problem 3. Does there exist a generalisation of Constraint Logic that explains the computational complexity of *Magic: The Gathering*?

Although the Halting problem is reducible to our construction, the fact that even evaluating a board is non-computable strongly suggests that the complexity of strategic play is greater than that. In particular, we conjecture:

▶ Conjecture 4. Playing *Magic: The Gathering* optimally is at least as hard as $\emptyset^{(\omega)}$.

Whether or not it is possible for there to be a real-world game whose computational complexity is strictly harder than $\emptyset^{(\omega)}$ is an interesting question on both a mathematical and a philosophical level. If not, then this conjecture would imply that *Magic: The Gathering* is as hard as it is possible for a real-world game to be.

5.2 Real-world Playability and Legality

While there are practical difficulties involved with correctly setting up the necessary board state, such as running out of space on your table, a sufficiently tenacious player could set up and execute this construction in a real-world tournament game of *Magic: The Gathering*. An example 60-card deck that is capable of executing this construction on the first turn of the game and which is legal in the competitive Legacy format can be seen in Table 3 in Appendix B.

With the correct draw, the deck uses **Ancient Tomb** ("tap: Add two colorless mana to your mana pool. Ancient Tomb deals 2 damage to you.") and three **Lotus Petal**s ("tap, Sacrifice Lotus Petal: Add one mana of any color to your mana pool. Play this ability as a mana source.") to play **Grim Monolith** ("Grim Monolith does not untap during your untap phase. T: Add three colorless mana to your mana pool. Play this ability as a mana source.") and **Power Artifact** ("Enchantment - Aura. Enchant artifact Enchanted artifact's activated abilities cost less to activate. This effect can't reduce the amount of mana an ability costs to activate to less than one mana.") and generate unlimited colourless mana, at which point **Staff of Domination** ("5, tap: Draw a card.") draws the rest of the deck and **Gemstone Array** ("2: Put a charge counter on Gemstone Array. Remove a charge counter from Gemstone Array: Add one mana of any color.") generates unlimited coloured mana. The deck casts most of the permanents immediately, and uses **Stolen Identity** ("Create a token that's a copy of target artifact or creature.") to make token copies of them (using **Memnarch** ("1UU: Target permanent becomes an artifact in addition to its other types.") first on the enchantments like **Cloak of Invisibility**). The tape is initialised with **Riptide Replicator** ("As Riptide Replicator enters the battlefield, choose a color and a creature type. Riptide Replicator enters the battlefield with X charge counters on it. 4, tap: Create an X/X creature token of the chosen color and type, where X is the number of charge counters on Riptide Replicator.") and **Capsize** ("Buyback 3 Return target permanent to its owner's hand."), **Reito Lanturn** ("3: Put target card in a graveyard on the bottom of its owner's library") allows repeated casting of the text-modification cards,

Reality Ripple ("Target artifact, creature, or land phases out.") sets the phase of the **Rotlung Reanimator**s and **Donate** ("Target player gains control of target permanent you control.") gives most permanents to Bob. Once the board is set up, **Reito Lantern** is also used to get Alice's library into the correct order. Finally, **Steely Resolve** ("As Steely Resolve enters the battlefield, choose a creature type. Creatures of the chosen type have shroud.") is cast, and then **Karn Liberated** and **Capsize** are used to exile all setup permanents and all cards from both players' hands so that neither player can interact with the operation of the Turing machine.

In addition to the *Comprehensive Rules* [16], play at sanctioned *Magic: The Gathering* tournaments is also governed by the *Tournament Rules* [17]. Some of these rules, most notably the ones involving slow play, may affect an individual's ability to successfully execute the combo due to concerns about the sheer amount of time it would take to manually move the tokens around to simulate a computation on a Turing machine. This would not be a concern for two agents with sufficiently high computational power, as the Tournament Rules also provide a mechanism called "shortcuts" for players to skip carrying out laborious loops if both players agree on the game state at the beginning and the end of the shortcut.

6 Conclusion

We have presented a methodology for embedding Rogozhin's (2, 18) universal Turing machine in a two-player game of *Magic: The Gathering*. Consequently, we have shown that identifying the outcome of a game of *Magic* in which all moves are forced for the rest of the game is undecidable. In addition to solving a decade-old outstanding open problem, in the process of arriving at our result we showed that *Magic: The Gathering* does not fit assumptions commonly made by computer scientists while modelling games. We conjecture that optimal play in *Magic* is far harder than this result alone implies, and leave the true complexity of *Magic* and the reconciliation of *Magic* with existing theories of games for future research.

References

1 David Auger and Oliver Teytaud. The frontier of decidability in partially observable recursive games. *International Journal of Foundations of Computer Science*, 2012.

2 Stella Biderman and Bjørn Kjos-Hanssen. Non-comparable natural numbers. Theoretical Computer Science Stack Exchange, 2018. URL: `https://cstheory.stackexchange.com/q/41384(version:2018-08-16)`.

3 Krishnendu Chatterjee and Rasmus Ibsen-Jensen. The complexity of deciding legality of a single step of Magic: The Gathering. In *22nd European Conference on Artificial Intelligence*, 2016.

4 Alex Churchill. *Magic: The Gathering* is Turing complete v5, 2012. URL: `https://www.toothycat.net/~hologram/Turing/`.

5 Alex Churchill et al. Magic is Turing complete (the Turing machine combo), 2014. URL: `http://tinyurl.com/pv3n2lg`.

6 Michael J. Coulombe and Jayson Lynch. Cooperating in video games? Impossible! Undecidability of team multiplayer games. In *9th International Conference on Fun with Algorithms*, 2018.

7 Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: algorithmic combinatorial game theory. In *26th Symp. on Mathematical Foundations in Computer Science*, pages 18–32, 2001.

8 Erik D. Demaine and Robert A. Hearn. Constraint logic: A uniform framework for modeling computation as games. In *2008 23rd Annual IEEE Conference on Computational Complexity*, pages 149–162, 2008.

9 Erik D. Demaine and Robert A. Hearn. *Games, Puzzles, and Computation.* CRC Press, 2009.

10 Alexander Esche. *Mathematical Programming and Magic: The Gathering.* PhD thesis, Northern Illinois University, 2018.

11 Eugenio Fortanely. Personal communication, 2018.

12 H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

13 Yurii Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

14 Colin D. Ward and Peter I. Cowling. Monte Carlo search applied to card selection in Magic: The Gathering. In *CIG'09 Proceedings of the 5th international conference on Computational Intelligence and Games*, pages 9–16, 2009.

15 Colin D. Ward, Peter I. Cowling, and Edward J. Powley. Ensemble determinization in Monte Carlo tree search for the imperfect information card game Magic: The Gathering. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 4, 2012.

16 Wizards of the Coast. Magic: The Gathering comprehensive rules, August 2018. URL: `https://magic.wizards.com/en/game-info/gameplay/rules-and-formats/rules`.

17 Wizards of the Coast. Magic: The Gathering tournament rules, August 2018. URL: `https://wpn.wizards.com/sites/wpn/files/attachements/mtg_mtr_21jan19_en.pdf`.

A How to Play *Magic: the Gathering*

In this section we provide a brief overview of the game and its rules, with a focus on what is necessary to understand the Turing machine construction. The full *Magic: the Gathering* Comprehensive Rules document [16] is over 200 pages of text and detailing them falls outside the purview of this paper.

A.1 An Introduction to *Magic*

Magic: the Gathering is a tabletop card game about magical combat. Each player brings their own deck of cards that they've chosen, called their *library*. On a player's turn, that player plays cards to cast spells, summon creatures, and use abilities of cards they've already played. When creatures die or when one-time effects are used, those cards are placed in a discard pile called the *graveyard*. Each player begins with 20 *life points* and once they are depleted that player loses the game; the main way of reducing the opponent's life total is to attack with creatures. A player can also lose the game if they attempt to draw from a library with no cards remaining.

Cards have various *types*. Card types include:

1. **Creature**. Creature cards are *permanents*, which means they are played onto the table (the *battlefield*) and remain in play. Creatures have two important statistics: their *power*, representing how much damage they can deal in combat, and their *toughness*, representing how much health they have or how far they are from dying. Standard notation for these uses a slash: a 3/2 creature has 3 power and 2 toughness. If a creature's toughness is ever zero or less, that creature *dies* and is put into the graveyard.
2. **Artifact** and **Enchantment**. These are also permanents. They do not engage in combat but have abilities that affect players or other permanents. Some enchantments are *Aura*s: these are attached to one other permanent or player and have an effect on that permanent or player. There are minor differences between artifacts and enchantments but they are not relevant to our construction.

3. **Instant** and **Sorcery**. These are cards that have a one-time effect and are then immediately put into the graveyard. They may cause effects that last until the end of the turn or which last indefinitely.
4. **Land**: Land cards remain on the battlefield. Usually they provide the resource known as *mana* which is used to play other cards. In our construction we have one land card but we ensure its controller cannot activate its ability.

Some cards have subtypes in addition to types. Aura is an example of a subtype of enchantments. All creatures have subtypes called *creature types* such as Goblin or Wizard that denote their race or class, and those creature types are used in various ways throughout the construction, in particular to track the symbols written onto the Turing tape.

A.2 Tokens

Some effects can create *tokens* on the battlefield, which are also permanents. This is crucial to the construction of a Turing tape potentially millions of cells long with a bounded number of cards. Tokens may be creatures, generally with no abilities, or they may be copies of other permanents such as enchantments or artifacts. Unless an effect specifies otherwise, tokens are treated exactly like cards of the same type while they are on the battlefield.

Tokens can only exist on the battlefield – if they ever leave the battlefield they cease to exist. If a creature token dies, it leaves the battlefield and goes to the graveyard (triggering any effects that watch for those conditions such as **Rotlung Reanimator**'s). However, it does not continue to exist in the graveyard.

A.3 Tapping

"Tapping" is a core mechanic in *Magic: the gathering* typically represented by turning a card 90 degrees. Being tapped is a binary state: a permanent is either tapped or untapped. At the beginning of each player's turn, the very first thing they do is untap all tapped permanents that they control. While there are a variety of ways that permanents can become tapped (some of which are used to set up the device), in the operation of the *Magic* Turing machine permanents will never become tapped. Tokens will be created either in a tapped or in an untapped state, and the difference between tapped and untapped tokens will control the state of the *Magic* Turing machine by controlling phasing.

A.4 Editing Card Text and Types

The Turing machine construction is only possible because certain *Magic* cards allow modification of the text of other cards, to change colours or creature types. The card **Artificial Evolution** reads "Change the text of target spell or permanent by replacing all instances of one creature type with another. The new creature type can't be Wall. *(This effect lasts indefinitely.)*" This card is vital to the construction, more so than any other.

Also crucial is one of several cards such as **Glamerdye** which read "Change the text of target spell or permanent by replacing all instances of one colour word with another". Similarly, we can change what colour a permanent is with **Prismatic Lace** ("Target permanent becomes the colour or colours of your choice").

For example, the card **Rotlung Reanimator** reads "Whenever Rotlung Reanimator or another Cleric dies, create a 2/2 black Zombie creature token". By casting two copies of **Artificial Evolution** replacing "Cleric" with "Aetherborn" and "Zombie" with "Sliver", and one copy of **Glamerdye** to replace "black" with "white", we can change **Rotlung**

Reanimator to read instead “Whenever Rotlung Reanimator or another *Aetherborn* dies, create a 2/2 *white Sliver* creature token”. This allows us to use creature types to track values throughout the computation, killing creature tokens with particular types and using **Rotlung Reanimator** as a conditional logic gate.

It is useful to add extra creature types to some creatures without changing their text box: this can be accomplished with **Olivia Voldaren**, who has the ability “Olivia Voldaren deals 1 damage to another target creature. That creature becomes a Vampire in addition to its other types.” We use **Artificial Evolution** to change **Olivia Voldaren** to add the creature type “Assembly-Worker” instead of “Vampire”: we will use the type Assembly-Worker to denote infrastructure creatures which will be rendered untargetable by spells.

It should be noted that all these edits only persist for as long as the permanent remains on the battlefield. If an edited permanent changes zone, such as going to the graveyard or the library, these edits are lost. This means that for cards the machine plays from players' hands, we cannot edit them; we need to work with their text as printed.

A.5 Abilities and the Stack

There are many different types of abilities that cards in *Magic: the Gathering* can have. The rules surrounding using abilities get rather complicated, but are crucial to understanding the mechanisms of the constructions in this paper. In this section, we restrict ourselves to explaining the bare minimum required to understand the construction.

Our construction is primarily concerned with *static abilities* and *triggered abilities*. Static abilities are abilities that are “always on” and modify the general rules of the game. For example, **Dread of Night** reads “White creatures get -1/-1.” This is a static ability of the **Dread of Night** permanent, affecting all white creatures and reducing their power and toughness by 1 each.

Triggered abilities begin with one of the words “When”, “Whenever” or “At”. **Rotlung Reanimator** has a triggered ability that reads “Whenever Rotlung Reanimator or another Cleric dies, create a 2/2 black Zombie creature token.” **Rotlung Reanimator** is how we will perform many of the functions of the Turing machine. It is a creature with two subtypes: Zombie and Cleric.

Whenever a spell is cast or an ability is activated or triggered, it is first put in a holding area known as the *stack*. When a spell or ability is on the stack, other players may add additional spells or abilities to the stack before the effect *resolves* (takes effect). The stack in *Magic* functions exactly like the data structure of the same name, with the spell or ability put on the stack first being carried out last and the spell or ability put on the stack last being carried out first.

The player whose turn it is is always first to get *priority*, which is permission to add new spells or abilities to the stack, followed by the player whose turn it isn't. Once both players decide to not use their priority to put a spell or ability on the stack, the top effect on the stack is popped and resolves.

Sometimes two triggered abilities will try to go on the stack at the same time. In this case, the order is determined by *Active Player, Nonactive Player (APNAP)* order. The active player is the one whose turn it is. Since this is the order the spells and abilities go on the stack, they will resolve in the reverse order (so the nonactive player's ability resolves first). If both effects are controlled by the same player, that player must choose the order to place them onto the stack. This poses a major limitation throughout the construction, as we aim to eliminate all choices by the players, so we cannot allow a player to control two effects that simultaneously trigger.

Although it is likely possible to simulate a Turing machine using the stack directly, our construction opts to take another tack.

A.6 Phasing

Phasing is an unusual ability some *Magic: the Gathering* cards have that is crucial to our construction. It allows a creature to be treated as if it doesn't exist – in particular, its triggered abilities won't trigger – but it stays on the battlefield, and so edits to its text by **Artificial Evolution** remain. Permanents with phasing toggle between two states each time their controller's turn begins: at the very beginning of a player's turn, all their phased-in permanents with phasing "phase out" (temporarily cease to exist) and all their phased-out permanents "phase in" (come back into existence).

A.7 Counters

There are many effects that can change the power and toughness of creatures. Some of these are temporary and last until the end of turn, while others are permanent. Permanent changes are denoted by *counters* placed on the creatures. In our construction, we will utilize +1/+1 and -1/-1 counters. +1/+1 counters increase the power and toughness of a creature each by 1, while -1/-1 counters decrease them. Pairs of +1/+1 and -1/-1 counters on the same creature cancel out.

A.8 The Structure of a Turn

Play in *Magic: the Gathering* consists of players alternately taking turns. Each turn is divided into phases, with each phase divided into steps such as the *upkeep step*. Many cards in *Magic* say something like "At the beginning of your upkeep..." or "At the beginning of each player's draw step..." At the beginning of each step and phase, the first thing done is always to check for such abilities and put them on the stack. During each of these phases, there is the option to cast spells and activate abilities, but some additionally have game actions players are required to take after all relevant effects have resolved.

The first phase of each turn is the *beginning phase*, which consists of the *untap step*, the *upkeep step*, and the *draw step*. During the untap step players first carry out any phasing effects, and then untap all permanents they control. There are no game actions during the upkeep step, and during the draw step the active player draws one card. Most of the action of the Turing machine happens in the beginning phase.

The second phase is the *first main phase*, where the bulk of the play occurs during a normal game, though nothing happens in the Turing machine. The third phase is the *combat phase*, where combat occurs, but this is not used in our construction. We will also have nothing relevant happen in the fourth phase (the *second main phase*) and only minimal effects during the final *end phase*.

B Tables

Table 1 Game state when the (2, 18) UTM begins.

Card	Controller	Changed text / details
29 Rotlung Reanimator	Bob	See Table 2
7 Xathrid Necromancer	Bob	See Table 2
29 Cloak of Invisibility	Alice	attached to Rotlung Reanimator
7 Cloak of Invisibility	Alice	attached to Xathrid Necromancer
Wheel of Sun and Moon	Alice	attached to Alice
Illusory Gains	Alice	attached to latest tape token
Steely Resolve	Alice	*Assembly-Worker*
2 Dread of Night	Alice	*Black*
Fungus Sliver	Alice	*Incarnation*
Rotlung Reanimator	Alice	*Lhurgoyf, black, Cephalid*
Rotlung Reanimator	Bob	*Lhurgoyf, green, Lhurgoyf*
Shared Triumph	Alice	*Lhurgoyf*
Rotlung Reanimator	Alice	*Rat, black, Cephalid*
Rotlung Reanimator	Bob	*Rat, white, Rat*
Shared Triumph	Alice	*Rat*
Wild Evocation	Bob	None
Recycle	Bob	None
Privileged Position	Bob	None
Vigor	Alice	None
Vigor	Bob	None
Mesmeric Orb	Alice	None
Ancient Tomb	Alice	None
Prismatic Omen	Alice	None
Choke	Alice	None
Blazing Archon	Alice	None
Blazing Archon	Bob	None

Table 2 Text of the Rotlung Reanimators and Xathrid Necromancers encoding the (2, 18) UTM.

Rogozhin's program				Card text				
q_1	1	c_2	Lq_1	Whenever an	Aetherborn	dies, create a	2/2 white	Sliver
q_1	$\overrightarrow{1}$	$\overleftarrow{1}_1$	Rq_1	Whenever a	Basilisk	dies, create a	2/2 green	Elf
q_1	$\overleftarrow{1}$	c_2	Lq_1	Whenever a	Cephalid	dies, create a	2/2 white	Sliver
q_1	$\overrightarrow{1}_1$	1	Rq_1	Whenever a	Demon	dies, create a	2/2 green	Aetherborn
q_1	$\overleftarrow{1}_1$	$\overrightarrow{1}_1$	Lq_1	Whenever an	Elf	dies, create a	2/2 white	Demon
q_1	b	$\overleftarrow{b}$	Rq_1	Whenever a	Faerie	dies, create a	2/2 green	Harpy
q_1	$\overrightarrow{b}$	$\overleftarrow{b}_1$	Rq_1	Whenever a	Giant	dies, create a	2/2 green	Juggernaut
q_1	$\overleftarrow{b}$	b	Lq_1	Whenever a	Harpy	dies, create a	2/2 white	Faerie
q_1	$\overrightarrow{b}_1$	b	Rq_1	Whenever an	Illusion	dies, create a	2/2 green	Faerie
q_1	$\overleftarrow{b}_1$	$\overrightarrow{b}_1$	Lq_1	Whenever a	Juggernaut	dies, create a	2/2 white	Illusion
q_1	b_2	b_3	Lq_2	Whenever a	Kavu	dies, create a tapped	2/2 white	Leviathan
q_1	b_3	$\overrightarrow{b}_1$	Lq_2	Whenever a	Leviathan	dies, create a tapped	2/2 white	Illusion
q_1	c	$\overrightarrow{1}$	Lq_2	Whenever a	Myr	dies, create a tapped	2/2 white	Basilisk
q_1	$\overrightarrow{c}$	$\overleftarrow{c}$	Rq_1	Whenever a	Noggle	dies, create a	2/2 green	Orc
q_1	$\overleftarrow{c}$	$\overrightarrow{c}_1$	Lq_1	Whenever an	Orc	dies, create a	2/2 white	Pegasus
q_1	$\overrightarrow{c}_1$	$\overleftarrow{c}_1$	Rq_2	Whenever a	Pegasus	dies, create a tapped	2/2 green	Rhino
q_1	$\overleftarrow{c}_1$	$HALT$		Whenever a	Rhino	dies, create a	2/2 blue	Assassin
q_1	c_2	$\overleftarrow{1}$	Rq_1	Whenever a	Sliver	dies, create a	2/2 green	Cephalid
q_2	1	$\overleftarrow{1}$	Rq_2	Whenever an	Aetherborn	dies, create a	2/2 green	Cephalid
q_2	$\overrightarrow{1}$	$\overleftarrow{1}$	Rq_2	Whenever a	Basilisk	dies, create a	2/2 green	Cephalid
q_2	$\overleftarrow{1}$	$\overrightarrow{1}$	Lq_2	Whenever a	Cephalid	dies, create a	2/2 white	Basilisk
q_2	$\overrightarrow{1}_1$	$\overleftarrow{1}_1$	Rq_2	Whenever a	Demon	dies, create a	2/2 green	Elf
q_2	$\overleftarrow{1}_1$	1	Lq_2	Whenever an	Elf	dies, create a	2/2 white	Aetherborn
q_2	b	b_2	Rq_1	Whenever a	Faerie	dies, create a tapped	2/2 green	Kavu
q_2	$\overrightarrow{b}$	$\overleftarrow{b}$	Rq_2	Whenever a	Giant	dies, create a	2/2 green	Harpy
q_2	$\overleftarrow{b}$	$\overrightarrow{b}$	Lq_2	Whenever a	Harpy	dies, create a	2/2 white	Giant
q_2	$\overrightarrow{b}_1$	$\overleftarrow{b}_1$	Rq_2	Whenever an	Illusion	dies, create a	2/2 green	Juggernaut
q_2	$\overleftarrow{b}_1$	$\overrightarrow{b}$	Lq_2	Whenever a	Juggernaut	dies, create a	2/2 white	Giant
q_2	b_2	b	Rq_1	Whenever a	Kavu	dies, create a tapped	2/2 green	Faerie
q_2	b_3	$\overleftarrow{b}_1$	Rq_2	Whenever a	Leviathan	dies, create a	2/2 green	Juggernaut
q_2	c	$\overleftarrow{c}$	Rq_2	Whenever a	Myr	dies, create a	2/2 green	Orc
q_2	$\overrightarrow{c}$	$\overleftarrow{c}$	Rq_2	Whenever a	Noggle	dies, create a	2/2 green	Orc
q_2	$\overleftarrow{c}$	$\overrightarrow{c}$	Lq_2	Whenever an	Orc	dies, create a	2/2 white	Noggle
q_2	$\overrightarrow{c}_1$	c_2	Rq_2	Whenever a	Pegasus	dies, create a	2/2 green	Sliver
q_2	$\overleftarrow{c}_1$	c_2	Lq_1	Whenever a	Rhino	dies, create a tapped	2/2 white	Sliver
q_2	c_2	c	Lq_2	Whenever a	Sliver	dies, create a	2/2 white	Myr

Table 3 60-Card Decklist to play the Turing machine in a Legacy tournament.

Card	Purpose	Card	Purpose
4 Ancient Tomb	Bootstrap	1 Rotlung Reanimator	Logic processing
4 Lotus Petal	Bootstrap	1 Cloak of Invisibility	Logic processing
4 Grim Monolith	Infinite mana device	1 Infest	Logic processing
4 Power Artifact	Infinite mana device	1 Cleansing Beam	Logic processing
4 Gemstone Array	Infinite mana device	1 Soul Snuffers	Logic processing
4 Staff of Domination	Draw rest of deck	1 Illusory Gains	Logic processing
1 Memnarch	Make token copies	1 Privileged Position	Logic processing
1 Stolen Identity	Make token copies	1 Steely Resolve	Logic processing
1 Artificial Evolution	Edit cards	1 Vigor	Logic processing
1 Olivia Voldaren	Edit cards	1 Fungus Sliver	Logic processing
1 Glamerdye	Edit cards	1 Dread of Night	Logic processing
1 Prismatic Lace	Edit cards	1 Wild Evocation	Forced play device
1 Donate	Edit card control	1 Wheel of Sun and Moon	Forced play device
1 Reality Ripple	Edit card phase	1 Shared Triumph	Infinite tape device
1 Djinn Illuminatus	Simplify setup	1 Xathrid Necromancer	Change state
1 Reito Lantern	Simplify setup	1 Mesmeric Orb	Change state
1 Claws of Gix	Simplify setup	1 Coalition Victory	Halting device
1 Riptide Replicator	Set up tape	1 Prismatic Omen	Halting device
1 Capsize	Set up tape	1 Choke	Halting device
1 Karn Liberated	Cleanup after setup	1 Recycle	Remove choices
1 Fathom Feeder	Cleanup after setup	1 Blazing Archon	Remove choices

Computational Fun with Sturdy and Flimsy Numbers

Trevor Clokie
University of Waterloo, Canada
trevor.clokie@uwaterloo.ca

Thomas F. Lidbetter
University of Waterloo, Canada
finnlidbetter@gmail.com

Antonio J. Molina Lovett
University of Waterloo, Canada
antonio@amolina.ca

Jeffrey Shallit
University of Waterloo, Canada
shallit@uwaterloo.ca

Leon Witzman
University of Waterloo, Canada
lwitzman@uwaterloo.ca

Abstract

Following Stolarsky, we say that a natural number n is *flimsy* in base b if some positive multiple of n has smaller digit sum in base b than n does; otherwise it is *sturdy*. We develop algorithmic methods for the study of sturdy and flimsy numbers.

We provide some criteria for determining whether a number is sturdy. Focusing on the case of base $b = 2$, we study the computational problem of checking whether a given number is sturdy, giving several algorithms for the problem. We find two additional, previously unknown sturdy primes. We develop a method for determining which numbers with a fixed number of 0's in binary are flimsy. Finally, we develop a method that allows us to estimate the number of k-flimsy numbers with n bits, and we provide explicit results for $k = 3$ and $k = 5$. Our results demonstrate the utility (and fun) of creating algorithms for number theory problems, based on methods of automata theory.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Theory of computation → Dynamic programming

Keywords and phrases sturdy number, flimsy number, context-free grammar, finite automaton, enumeration

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.10

Related Version The full paper is available at `https://arxiv.org/abs/2002.02731`.

Supplementary Material Implementations of our algorithms can be found in the GitHub repository `https://github.com/FinnLidbetter/sturdy-numbers`.

Acknowledgements We would like to thank Kenneth Stolarsky and the referees for helpful comments.

1 Introduction

Let $s_b(n)$ denote the sum of the digits of n, when expressed in base b. Thus, for example, $s_2(9) = 2$. A number n is said to be *k-flimsy in base b* if there exists a positive integer k such that $s_b(kn) < s_b(n)$. Any such k, if one exists, is called a *flimsy witness* for n. If n is k-flimsy for some k, it is said to be *flimsy*. If there is no such k, then n is said to be *sturdy in base b*. For example, 7 is sturdy in base 2, while 13 is flimsy, because $s_2(13) = 3 > 2 = s_2(5 \cdot 13)$. Thus 5 is a flimsy witness for 13. In this paper we examine the computational aspects of sturdy and flimsy numbers.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 10; pp. 10:1–10:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sturdy and flimsy numbers were introduced by Stolarsky in 1980 [28]. For other papers on the topic, see [25, 10, 8, 6].

Many of the sequences we discuss appear in the *On-Line Encyclopedia of Integer Sequences* [27]. For example, the base-2 sturdy numbers form sequence A125121 in the OEIS, while the base-2 sturdy primes form sequence A143027. The base-2 flimsy numbers form sequence A005360, while the base-2 flimsy primes form sequence A330696. The base-10 sturdy numbers form sequence A181862, while the base-10 sturdy primes form sequence A181863. Sequence A086342 gives the value of $\min_{k\geq 1} s_2(kn)$, while sequence A143069 gives $\operatorname{argmin}_{k\geq 1} s_2(kn) = \min\{k : s_2(kn) = \min_{k\geq 1} s_2(kn)\}$.

The goal of this paper is to examine the algorithmic aspects of sturdy and flimsy numbers. The outline of the paper is as follows. In Section 2, we prove some basic properties of digit sums of multiples. In Section 3, we give a criterion for determining if a number is flimsy, and use it to find two previously unknown sturdy primes.

Next, we turn to algorithms for sturdy and flimsy numbers. A priori it is not immediately clear that it is even decidable whether a given n is flimsy or sturdy. Indeed, in a recent paper by Elsholtz [11], he asks, “How can one algorithmically find a “sparse” representation of a multiple of p?”

More precisely, there are four computational problems worthy of study:

1. Given a positive integer n, decide whether it is sturdy in base b.
2. Compute $\operatorname{swm}_b(n) := \min_{k\geq 1} s_b(kn)$. This is the *smallest weight of a multiple*; the smallest digit sum of a multiple of n; if n is sturdy, then $\operatorname{swm}_b(n) = s_b(n)$.
3. Compute $\operatorname{msw}_b(n) := \operatorname{argmin}_{k\geq 1} s_b(kn)$. This is the *minimum sum witness*; the smallest k such that kn achieves its minimum digit sum; if n is sturdy, then $\operatorname{msw}_b(n) = 1$.
4. Given that n is flimsy, determine $\operatorname{mfw}_b(n) := \min\{k : s_b(kn) < s_b(n)\}$. This is the *minimal flimsy witness.*

A table of these functions is given in Table 1. Here the column labeled “char” is F if the number is flimsy and S if it is sturdy.

In Sections 4–7, we discuss algorithms to solve these problems. The fastest, based on automata theory, shows that we can check whether a number n is sturdy in $O(n)$ time. Section 10 gives our computational results achieved with our algorithms.

In Section 11 we give an application of automata to help characterize the flimsy numbers with a fixed number of 0's.

Finally, in Section 12, we turn to estimating the number of k-flimsy numbers with n bits. We use techniques from formal language theory to solve the problem.

2 Basic properties

In this section, we prove some of the basic properties of digit sums of multiples.

We start with some notation. For $n \geq 0$, we define $(n)_b$ to be the base-b representation of n, starting with the most significant digit. If x is a string, we define $[x]_b$ to be the integer that x represents when interpreted in base b. If b is fixed, we define $\overline{x}$ to be the base-b complement of x, that is, the string where each digit d in x is replaced by $b - 1 - d$.

▶ **Theorem 1.** *Let $b \geq 2$ be an integer, and t be a positive divisor of b. Then for all integers $n, r \geq 1$, there exists a positive integer j such that $s_b(jn) = r$ if and only if there exists a positive integer k such that $s_b(ktn) = r$.*

Proof. For one direction, take $j = kt$.

For the other direction, assume that there exists $j \geq 1$ such that $s_b(jn) = r$. Let $k = bj/t$. Then $s_b(ktn) = s_b(bjn) = s_b(jn) = r$. ◀

Table 1 Table of sturdy and flimsy numbers.

n	char	swm(n)	msw(n)	mfw(n)	n	char	swm(n)	msw(n)	mfw(n)
3	S	2	1	-	5	S	2	1	-
7	S	3	1	-	9	S	2	1	-
11	F	2	3	3	13	F	2	5	5
15	S	4	1	-	17	S	2	1	-
19	F	2	27	27	21	S	3	1	-
23	F	3	3	3	25	F	2	41	41
27	F	2	19	3	29	F	2	565	5
31	S	5	1	-	33	S	2	1	-
35	S	3	1	-	37	F	2	7085	7085
39	F	3	7	7	41	F	2	25	25
43	F	2	3	3	45	S	4	1	-
47	F	3	11	3	49	S	3	1	-
51	S	4	1	-	53	F	2	1266205	5
55	F	3	7	3	57	F	2	9	9
59	F	2	9099507	3	61	F	2	17602325	5
63	S	6	1	-	65	S	2	1	-
67	F	2	128207979	128207979	69	S	3	1	-
71	F	3	119	119	73	S	3	1	-
75	S	4	1	-	77	F	3	5	5
79	F	3	13	7	81	F	2	1657009	1657009
83	F	2	26494256091	395	85	S	4	1	-
87	F	3	3	3	89	S	4	1	-
91	F	3	3	3	93	S	5	1	-
95	F	3	5519	3	97	F	2	172961	172961
99	F	2	331	11	101	F	2	11147523830125	365
103	F	3	5	5	105	S	4	1	-
107	F	2	84179432287299	3	109	F	2	2405	5
111	F	3	591	3	113	F	2	145	145
115	F	3	571	9	117	F	4	5	5
119	F	3	71	3	121	F	2	297758653049289	9
123	F	4	19	3	125	F	2	9007199254741	5
127	S	7	1	-	129	S	2	1	-

We now show that in order to compute swm_b, it suffices to consider only those arguments relatively prime to b.

▶ **Corollary 2.** *Write the prime factorization of n as $\prod_{1\le i\le t} p_i^{e_i}$, and define $g = \prod_{p_i \mid b} p_i^{e_i}$. Then $\mathrm{swm}_b(n) = \mathrm{swm}_b(n/g)$, and $\gcd(b, n/g) = 1$.*

Proof. Let p be any prime dividing both b and n. From Theorem 1, we see that $\mathrm{swm}_b(n) = \mathrm{swm}_b(n/p)$. By repeatedly applying this observation, and replacing n with n/p, we can remove from n all primes dividing both b and n, while maintaining the same value of swm_b. At the end, the resulting n/g is relatively prime to b. ◀

▶ **Theorem 3.** *There exists $j \ge 1$ such that $s_b(jn) = t$ if and only if there exist t* distinct *powers of b that sum to a multiple of n.*

Proof. By Corollary 2, we may assume that n is coprime with b.

In such cases, b has finite order, say ν, modulo n. Suppose $\sum_{i=0}^{\nu-1} c_i b^i \equiv 0 \pmod{n}$ where each $c_i \ge 0$ and $\sum_{i=0}^{\nu-1} c_i = t$. Then $\sum_{i=0}^{\nu-1} \sum_{j=0}^{c_i-1} b^{j\nu+i} \equiv 0 \pmod{n}$, and this sum consists of distinct powers of b. ◀

Empirical evidence suggests that if $b = 2$ and $\mathrm{swm}_b(n) = t$, then for all $i \geq 0$, some multiple of n has digit sum $t + i$. However, the analogous result is false for $b = 3$. For example, $\mathrm{swm}_3(13) = 3$, but no multiple of 13 has digit sum 4. These observations are explained in the following theorem.

▶ **Theorem 4.** *Suppose j, n are positive integers such that $s_b(jn) = t$. Then for all $r \geq 0$, there exists $k \geq 1$ such that $s_b(kn) = t + r(b-1)$.*

Proof. Assume $s_b(jn) = t$ for some $t \geq 1$. Then from Theorem 3 we know that $\sum_{i=1}^{t} b^{m_i} \equiv 0 \pmod{n}$ for some strictly increasing m_i and (replacing j by bj if needed) we can assume $m_t \geq 1$. Now replace the high-order bit b^{m_t} in this sum with the sum of b terms $b^{\nu+m_t-1} + b^{2\nu+m_t-1} + \cdots + b^{(b-1)\nu+m_t-1}$, where ν is the order of b, modulo n. This has the effect removing 1 bit, while adding b additional bits, and each of the b new terms is congruent to $b^{m_t-1} \pmod{n}$. So we have found another multiple of n with digit sum $t + b - 1$. We can repeat this transformation any number of times. ◀

▶ Remark 5. The result is optimal. Since $b - 1$ divides the digit sum of any multiple of $b - 1$, there is no $k \geq 1$ satisfying $b - 1 < s_b(k(b-1)) < 2(b-1)$.

3 Infinite classes of sturdy numbers

We first give a criterion for deciding whether a number is flimsy. This shows that Problem 1 on our list, determining whether a given positive integer is sturdy, is decidable.

▶ **Theorem 6.** *Let n, b, j be positive integers, $b \geq 2$ such that n divides $b^j - 1$. Then n is flimsy in base b if and only if $s_b(kn) < s_b(n)$ for some k satisfying $1 \leq k \leq \frac{b^j-1}{n}$.*

Proof. One direction is easy: if $s_b(kn) < s_b(n)$ for some k, then n is flimsy in base b.

For the other direction, suppose n is flimsy, but $s_b(kn) \geq s_b(n)$ for all k with $1 \leq k \leq \frac{b^j-1}{n}$. Let k' be the smallest positive integer such that $s_b(k'n) < s_b(n)$. By assumption $k'n \geq b^j$, and so we can write $k'n = cb^j + d$ for uniquely-determined $c \geq 1$ and $0 \leq d < b^j$. Since $b^j \equiv 1 \pmod{n}$, it follows that $cb^j + d \equiv c + d \equiv 0 \pmod{n}$. Then $c + d = fn < cb^j + d = k'n$ for some integer f with $1 \leq f < k'$. Thus $s_b(k'n) = s_b(cb^j + d) = s_b(c) + s_b(d) \geq s_b(c + d) = s_b(fn) \geq s_b(n)$, achieving the desired contradiction. ◀

▶ Remark 7. Since $j \leq \varphi(n)$, this together with Theorem 1 shows that sturdiness is reduced to a finite search. The result for $b = 10$ was observed by Phedotov [20].

We applied Theorem 6 to known prime factors of composite Mersenne numbers [29] and found

$$579126141132756490877721 = \frac{2^{83} - 1}{167}$$

and

$$1035079443105516238671861923746823 4569 = \frac{2^{131} - 1}{263}$$

as previously unknown sturdy primes in base 2.

▶ **Corollary 8.** *If b, j are positive integers, with $b \geq 2$, then $\frac{b^j-1}{m}$ is sturdy in base b for every positive m dividing $b - 1$.*

Proof. Let k be an integer with $1 \le k \le m$. Then we have

$$s_b\left(k\frac{b^j-1}{m}\right) = s_b\left(\frac{k(b-1)}{m}\sum_{i=0}^{j-1} b^i\right) = kj\frac{b-1}{m} \ge j\frac{b-1}{m},$$

where we have used the fact that $k \le m$. The result now follows by Theorem 6. ◀

▶ **Theorem 9.** *Let n be sturdy in base $b \ge 2$, with n dividing $b^j - 1$ for some $j \ge 1$. Fix a positive integer r, and define $m = n\left(\frac{b^{rj}-1}{b^j-1}\right)$. Then m is sturdy in base b.*

Proof. Let $x = (n)_b$. Observe that $m = [(x\,0^{j-|x|})^{r-1}\,x]_b$, so $s_b(m) = rs_b(n)$. If $1 \le k \le \frac{b^j-1}{n}$, then $(km)_b$ consists of r copies of $(kn)_b$ concatenated, separated by some number of 0's. So $s_b(km) = rs_b(kn) \ge rs_b(n) = s_b(m)$. The result now follows by Theorem 6. ◀

We can now get a generalization of a theorem of Stolarsky [28, Thm. 2.1].

▶ **Corollary 10.** *Let $e, k, r \ge 1$, and $b \ge 2$. Define $n = [(1^k\,0^{(e-1)k})^{r-1}\,1^k]_b = \left(\frac{b^k-1}{b-1}\right)\left(\frac{b^{rek}-1}{b^{ek}-1}\right)$. Then n is sturdy in base b.*

Proof. Note that $\frac{b^k-1}{b-1}$ is sturdy in base b by Corollary 8. Additionally, $b^k - 1$ divides $b^{ek} - 1$. The result now follows by Theorem 9. ◀

The next theorem gives another infinite class of sturdy numbers.

▶ **Theorem 11.** *Fix $b \ge 2$. Let n be a positive integer, and x be the base-b representation of n. Then every integer with base-b representation of the form $x\,(b-1)^i\,\overline{x}$, where $i \ge 0$ and $\overline{x}$ is the base-b complement of x, is sturdy in base b.*

Proof. Suppose $y = x\,(b-1)^i\,\overline{x}$ for some $i \ge 0$. Then $[y]_b + n = nb^{|x|+i} + b^{|x|+i} - 1$. Then $[y]_b = (n+1)(b^{|x|+i}-1)$. Observe that $s_b(b^{|x|+i}-1) = (|x|+i)(b-1) = s_b([y]_b)$. Furthermore, $b^{|x|+i} - 1$ is sturdy in base b by Corollary 8. Then for every positive integer k we have $s_b(k[y]_b) = s_b(k(n+1)(b^{|x|+i} - 1)) \ge s_b(b^{|x|+i} - 1) = s_b([y]_b)$. ◀

▶ **Corollary 12.** *Let $b \ge 2$ be an integer, and m be a positive integer dividing $b - 1$. Then $\frac{(b^n-1)^2}{m}$ is sturdy in base b for all $n \ge 1$.*

Proof. Suppose m divides $b - 1$. Then we have

$$\begin{aligned}
\frac{(b^n-1)^2}{m} &= \frac{b^n-1}{m}b^n - \frac{b^n-1}{m}\\
&= \frac{b^n-1}{m}b^n - b^n + b^n - \frac{b^n-1}{m}\\
&= \left(\frac{b^n-1}{m} - 1\right)b^n + b^n - \frac{b^n-1}{m}\\
&= \left(\frac{b^n-1}{m} - 1\right)b^n + (b^n-1) - \left(\frac{b^n-1}{m} - 1\right),
\end{aligned}$$

which has base-b representation $x\overline{x}$ where $x = \left(\frac{b^n-1}{m} - 1\right)_b$. Then by Theorem 11, $\frac{(b^n-1)^2}{m}$ is sturdy in base b. ◀

In the rest of this paper we are almost exclusively concerned with the case $b = 2$, and so from now on we omit the subscripts on the functions $\text{msw}, \text{swm}, \text{mfw}$, and use the terms *flimsy* or *sturdy* without further elaboration. In this case $s_2(n)$ equals the number of 1's in the binary representation of n, also known as the *Hamming weight* of n.

4 Algorithms when $\mathrm{swm}(n)$ is small

As we will see in Section 5, for general n we can determine whether n is sturdy in $O(n)$ time. We call this a linear-time algorithm.[1] Therefore, it is of interest to see when this can be improved.

If $\mathrm{swm}(n)$ is small, this fact can be verified efficiently in some cases. This is particularly relevant in the case where n is prime because, according to a recent result of Elsholtz [11], almost all primes p have $\mathrm{swm}(p) \leq 7$. Furthermore, we know from results of Hasse [14] and Odoni [18] that a positive proportion of all primes satisfy $\mathrm{swm}(p) = 2$; asymptotically, this fraction is 17/24. For general n, however, the situation is different: the set of n for which $\mathrm{swm}(n) = 2$ has density 0; see the results of Moree in [21, Appendix B].

4.1 The case $\mathrm{swm}(n) = 2$

If $\mathrm{swm}(n) = 2$, then $n \cdot \mathrm{msw}(n) = 2^k + 1$ for some integer $k \geq 1$. Hence $n \,|\, 2^k + 1$, and so -1 belongs to the subgroup generated by 2 (mod n). We can decide if -1 belongs to the subgroup generated by 2 (mod n) by using an algorithm for the discrete logarithm problem. For example, the baby-step giant-step algorithm can be used to find k such that $2^k \equiv -1 \pmod{n}$, if such a k exists, with time complexity $O(\sqrt{n} \log n)$ [26]. If the factorization of n is known, this running time can be substantially improved.

4.2 The case $\mathrm{swm}(n) = 3$

If $\mathrm{swm}(n) = 3$, then $n \cdot \mathrm{msw}(n) = 2^k + 2^\ell + 1$ for some integers $k > \ell \geq 1$. It follows that $2^k + 2^\ell \equiv -1 \pmod{n}$, which means that we are dealing with a 2-SUM problem. This can be solved in $O(n \log n)$ time using sorting and binary search. (Briefly, compute a table of powers of 2, mod n; sort them in ascending order, and then for each power 2^k use binary search to see if there is an ℓ such that $2^\ell \equiv -1 - 2^k \pmod{n}$.) Although this does not beat our $O(n)$ algorithm given below asymptotically, in many cases it will run more quickly because of the simplicity of the operations. This is particularly true if the subgroup generated by 2 (mod n) is small.

5 A dynamic programming algorithm

In this section we show how to check whether n is sturdy using dynamic programming.

By Corollary 2, we can restrict our attention to the case where n is odd. In this case, the powers of two $P_n = \{2^i : i \geq 0\}$ form a cyclic subgroup of $(\mathbb{Z}/(n))^*$, the multiplicative group of integers relatively prime to n. Define $\nu = \mathrm{ord}_2\, n = |P_n|$, the order of 2 in the group $(\mathbb{Z}/(n))^*$. Hence, to find a positive multiple of n whose binary expansion contains exactly k 1's, it suffices to find an appropriate linear combination of k elements of P_n (counted with repetition) that sums to 0 (mod n). More precisely, we need to find non-negative integers $a_1, a_2, \ldots, a_i$ and distinct elements $e_1, e_2, \ldots, e_i \in P_n$ such that

$$a_1 e_1 + \cdots + a_i e_i \equiv 0 \pmod{n}$$
$$a_1 + a_2 + \cdots + a_i = k,$$

[1] Strictly speaking, the usage "linear-time" in the context of algorithms on integers would usually mean an algorithm that runs in $O(\log n)$ time. But since no algorithm is this efficient, we stray from the common usage for brevity.

for integers $k \geq 1$. This is the kind of problem that dynamic programming is well-suited for. To restrict the amount of work required in a dynamic programming algorithm for this we make use of the following lemma.

▶ **Lemma 13.** *For an integer base $b \geq 2$ let $P_{b,n} = \{b^i \bmod n : i \in \mathbb{N}\}$ and suppose that $e_1, e_2, \ldots, e_m$ are the distinct elements of $P_{b,n}$. If there exist non-negative integers $a_1, a_2, \ldots, a_m$ such that $a_1e_1+a_2e_2+\cdots+a_me_m \equiv 0 \pmod{n}$ and $a_1+\cdots+a_m = k$, then there exist non-negative integers $c_1, c_2, \ldots, c_m < b$ and $l \leq k$ such that $c_1e_1 + c_2e_2 + \cdots + c_me_m \equiv 0 \pmod{n}$ and $c_1 + \cdots + c_m = l$.*

Proof. Suppose we have non-negative integers $a_1, a_2, \ldots, a_m$ such that $a_1e_1 + a_2e_2 + \cdots + a_me_m \equiv 0 \pmod{n}$ and $a_1+\cdots+a_m = k$. If we have $a_1, a_2, \ldots, a_m < b$ then we are done. So suppose that there is some i such that $a_i \geq b$. Let j be the integer such that $be_i \equiv e_j \pmod{n}$. Then we can take

$$\left(\sum_{r=1, r\neq i, r\neq j}^{m} a_r e_r \right) + (a_i - b)e_i + (a_j + 1)e_j \equiv 0 \pmod{n},$$

giving

$$\left(\sum_{r=1, r\neq i, r\neq j}^{m} a_r \right) + (a_i - b) + (a_j + 1) = a_1 + \cdots + a_m - b + 1 = k - b + 1 < k.$$

Setting $a_i := a_i - b$ and $a_j := a_j + 1$ and $k := k - b + 1$, we can repeat this argument until $a_1, \ldots, a_m < b$. ◀

Let us start with determining whether n is sturdy. It suffices to solve the problem of the previous paragraph for $1 \leq k < s_2(n)$. The idea is that we will fill in the entries of a 3-dimensional boolean array x with the following meaning: the entry $x[i, j, r]$ is `true` if and only if the integer j has a representation as a sum of $i \geq 1$ powers of 2, using as summands only the first r elements of the set P_n without repetition. We fill in the array x in increasing order of r.

For initialization, we set all elements of x to `false`, except that we set $x[0, 0, r]$ to `true` for $0 \leq r \leq \nu$.

To solve the remaining three problems, we need to record more information than just the ability to represent j as a sum of powers of 2. The integer array $y[i, j, r]$ is used to record the smallest integer congruent to $j \pmod{n}$ that is the sum of exactly i powers of 2 (without repetition), using only the first r elements of the set P_n.

In the pseudocode that follows, the scope of loops is indicated by the indentation.

```
minrep(n)   { assumes n odd and at least 3 }

sumd := sumdig(2,n);   {sum of base-2 digits of n}

{make a table of powers of 2}
b := 1;
a := 0;
repeat
   b := (2*b) mod n;
   a := a+1;
```

```
until
   b = 1;
ord2 := a;   { the order of 2 mod n }
power2 := array[0..ord2-1] of integer;
for m := 0 to ord2-1 do
   power2[m] := b;
   b := (2*b) mod n;

{ the intent is that x[i,j,r] = true, if j (mod n) has a representation
as a sum of exactly i powers of 2, using only the first r elements of
power2 (without repetition), and false otherwise.
y[i,j,r] = smallest integer congruent to j (mod n)
representable by the sum of exactly i powers of 2,
using only the first r elements of power2 (without repetition) }

x := array[0..sumd-1, 0..n-1, 0..ord2] of boolean;
y := array[0..sumd-1, 0..n-1, 0..ord2] of integer;

{ initialize }

for r := 0 to ord2 do
     for i := 1 to sumd-1 do
         for j := 0 to n-1 do
            x[i,j,r] := false;
            y[i,j,r] := infinity;
    x[0,0,r] := true;
    y[0,0,r] := 0;

{ fill in table }

for r := 1 to ord2 do   {consider summand 2^{r-1} mod p}
   for j := 0 to n-1 do      { check position j }
       for i := 1 to sumd-1 do     {fill in level i of the array}
            x[i,j,r] := x[i,j,r-1];
            y[i,j,r] := y[i,j,r-1];
            {check if we can use 2^{r-1} }
            if x[i-1, (j-power2[r-1]) mod n, r-1] then
                x[i,j,r] := true;
                y[i,j,r] := min(y[i,j,r],
                y[i-1, (j-power2[r-1]) mod n, r-1] + 2^{r-1});

sturdy := true;
for i := 2 to sumd-1 do
    sturdy := sturdy and x[i,0,ord2];

if (sturdy) then
    print("swm(n) = ",sumd);
    print("msw(n) = ",1);
```

```
else
    i := 1;
    while (not x[i,0,ord2]) do i := i+1;
    print("swm(n) = ",i);
    print("msw(n) = ",y[i,0,ord2]/n);
    mfw := infinity;
    while (i < sumd) do
        mfw := min(mfw, y[i,0,ord2]);
        i := i+1;
    print("mfw(n) = ",mfw);

end;
```

Our dynamic programming algorithm has three nested loops, which gives a running time of $O(\nu \cdot n \cdot s_2(n))$. Since $\nu = \text{ord}_n 2$ could be as large as $n-1$, and $s_2(n)$ could be as big as $\log_2 n$, this gives a worst-case running time of $O(n^2 \log n)$, where we are measuring the run time in terms of RAM operations on integers of size about n. This means that this algorithm will only be feasible for integers smaller than about 10^7.

6 An algorithm based on finite automata

In this section we provide a different, much faster algorithm for checking sturdiness, based on finite automata.

The idea is simple. It is easy to create a deterministic finite automaton (DFA) accepting the binary representations of the positive integers divisible by n. Such an automaton has n states [1] and exactly one final state. Next, we can easily construct a DFA A_t accepting those strings starting with a 1 and having at most t ones. Using the standard “direct product” construction [15, pp. 59–60], we can construct a DFA M_t of $(t+2)n$ states for the intersection of these two languages; it has exactly $t+1$ final states $f_0, f_1, \ldots, f_t$ corresponding to positive integers divisible by n with $0, 1, \ldots, t$ 1’s respectively. Then some multiple of n has at most t 1’s iff M_t accepts at least one string. We can test this condition (and even find the lexicographically least string accepted) using breadth-first search to decide if some f_i for $0 \leq i \leq t$ is reachable from the start state of M_t, in linear time in the size of M, so in $O((t+2)n)$ time.

By choosing $t = s_2(n) - 1$ we can determine if n is sturdy in $O(n \log n)$ steps. Similarly, by allowing the breadth-first search to run to completion and keeping track of the least string in radix order used to reach each state, we can recover $\text{swm}(n)$, $\text{msw}(n)$, and $\text{mfw}(n)$ by examining each of the final states for whether or not they were visited in the search and looking at the least string in radix order used in each case. More precisely, the value of $\text{swm}(n)$ is the least integer i such that final state f_i in M_t is reached in the breadth-first search, or $s_2(n)$ if no final state is reached. The value of $\text{msw}(n)$ is the least string in radix order used to reach $f_{\text{swm}(n)}$ interpreted as an integer and divided by n, or 1 if n is sturdy. The value of $\text{mfw}(n)$, if it is defined, is the least string in radix order among all such strings used to reach a final state, interpreted as an integer and divided by n. To avoid needing to store the representation of large integers, we instead store the exponents of the current power of 2 and a pointer to the previous power. From this linked list we can reconstruct the appropriate number.

▶ **Theorem 14.** *We can decide whether n is sturdy $O(n \log n)$ steps. In the same time bound we can compute* $\mathrm{swm}(n)$ *and* $\mathrm{msw}(n)$*. If n is flimsy, we can compute* $\mathrm{mfw}(n)$ *in the same time bound.*

This algorithm is practical for n up to about 10^{10}. The main constraint is likely to be space and not time.

7 Improving the automaton-based algorithm

With a small modification to this idea of using a breadth-first search on the graph defined by automaton M, we can make further improvements to the time complexity. Consider the deterministic finite automaton M_n accepting the binary representations of the positive integers divisible by n. We then define a directed graph G_n with vertices given by the states of M_n and directed, weighted edges given by the transitions of M_n where transitions on the symbol 0 are given an edge weight of 0 in G_n and transitions on the symbol 1 are given an edge weight of 1 in G_n. We augment G_n with one additional vertex, v_s, with a single outgoing edge of weight 1 to the vertex corresponding to the state reached when M_n reads any input of the form 0^*1. If v_f is the vertex corresponding to the accepting state in M_n, then there is a path from v_s to v_f of weight k if and only if there is a non-zero multiple of n with Hamming weight k. The shortest path problem on a graph $G = (V, E)$ with edge weights in $\{0, 1\}$ can be solved in time $O(|V| + |E|)$ using a variation of the breadth-first search algorithm. In place of the queue used in a standard breadth-first search, we use a double ended queue. We process a node by traversing incident edges of weight 0 and pushing the nodes reached to the front of the queue if they have not been processed already. Edges of weight 1 are also traversed, but the nodes reached are pushed to the back of the queue provided that they have not been processed already. After a node has been processed, the next node at the front of the queue is dequeued and processed if it has not been processed already, otherwise it is just discarded. The depth of the search can be tracked as in a standard breadth-first search. Thus we achieve the following improvement.

▶ **Theorem 15.** *We can test if n is sturdy in $O(n)$ steps.*

From this approach we are still able to construct an example of a multiple of n achieving the minimum Hamming weight over all multiples of n. It is simply a matter of maintaining the path used in the breadth-first search algorithm finding the shortest path from v_s to v_f in G_n. However, there is no guarantee that this is the least multiple of n with this property. To find the least multiple we can use the linear-time algorithm to first determine the minimum Hamming weight. For minimum Hamming weight k, we take the direct product of automaton M_n accepting the base-2 representations of all multiples of n and the automaton accepting all binary strings with exactly k 1's. A breadth-first search on this product automaton gives the least non-zero multiple of n with Hamming weight k. This second breadth-first search has worst case time complexity $O(n \log n)$, giving overall complexity $O(n \log n)$ for finding the least non-zero multiple of n having the minimum Hamming weight over all non-zero multiples of n.

8 Another breadth-first search approach

We can take advantage of Lemma 13 to evaluate sturdiness and compute swm and msw using a breadth-first search on a different graph structure. As before, to test the sturdiness of an integer $n \geq 3$, we construct an $(n + 1)$-vertex graph with n of the vertices representing the distinct residue classes modulo n, which we will refer to as $[0], [1], \ldots, [n-1]$, and one special vertex, v_0, corresponding to the number 0. The graph contains a directed edge from vertex $[x]$ to vertex $[y]$ if and only if $x + 2^j \equiv y \pmod{n}$ for some integer $j \geq 0$. Similarly, there is an edge from v_0 to $[y]$ if and only if $2^j \equiv y \pmod{n}$. Hence each vertex has out-degree $\nu = \operatorname{ord}_n 2$. The idea of this construction is to treat traversing an edge from $[x]$ to $[x + 2^j]$ as choosing to use the jth power of 2 as a summand in a summation to a value congruent to 0 modulo n. Thus, to compute $\operatorname{swm}(n)$ we are looking for the length of the shortest path from v_0 to $[0]$ and this can be found via a breadth-first search. Furthermore, by keeping track of the smallest sum required to reach each state, we can also recover $\operatorname{msw}(n)$ from such a breadth-first search. Rather than running the breadth-first search to completion, we can terminate as soon as we reach a depth equal to $s_2(n)$, as we will know by then whether or not n is sturdy.

With this approach we can take advantage of the structure of the graph to speed up testing for sturdiness. If during the breadth first search we visit a node $[x]$ such that $[n - x]$ has already been visited, then since the length of the shortest path from $[x]$ to $[0]$ is equal to the length of the shortest path from v_0 to $[n - x]$ either we will know that n is not sturdy, or that it is not necessary to continue searching from $[x]$. This greatly improves the efficiency of the testing for sturdiness.

The complexity of this approach, for evaluating sturdiness, and computing $\operatorname{swm}(n)$ and $\operatorname{msw}(n)$ is $O(n^2)$ since we are performing a breadth-first search on a graph with $n+1$ vertices each with $\nu = O(n)$ outgoing edges. In practice this approach seems to perform much better than our naive $O(n^2)$ upper bound would suggest, especially in testing for sturdiness, due to the early exit conditions.

9 Running time comparison

To demonstrate how these algorithms behave in practice, we compiled timing information for each of the approaches and each of the four functions of interest for consecutive integers starting from 1. Each of the algorithms are implemented as described above. However, before applying each algorithm the baby-step giant-step algorithm, as in Section 4.1, is used to exit faster in those cases where $\operatorname{swm}(n) = 2$. Running times for the mfw function with the `order_deg_bfs` algorithm are not given because there does not seem to be a natural approach for using this idea to evaluate mfw. The computations producing the given running times were performed on macOS Catalina version 10.15.2 on a 2.3 GHz Intel Core i5 processor. Implementations of our algorithms can be found in the GitHub repository
`https://github.com/FinnLidbetter/sturdy-numbers`.

Table 2 Running time in milliseconds for each of the algorithms to evaluate the functions for all values of n (odd and even) between 1 and 2000 inclusive.

Algorithm	`is_sturdy`	swm	msw	mfw
`dp`	22836	2667063	5675228	5167556
`aut`	1042	1050	1473	1430
`order_deg_bfs`	322	1646	4339	—
`bfs01`	224	226	650	1416

In Tables 2 and 3, the algorithmic approaches are named according to the commands used in the program-runner in the GitHub repository. Here, the `dp` algorithm refers to the dynamic programming approach described in Section 5, the `aut` algorithm refers to the automaton-based approach described in Section 6, the `bfs01` algorithm refers to the improved automaton-based approach described in Section 7, and the `order_deg_bfs` algorithm refers to the alternative breadth-first search approach described in Section 8.

Table 3 Running time in milliseconds for the algorithms to evaluate the functions for all values of n (odd and even) between 1 and 10000 inclusive. The dynamic programming algorithm was not included because it was not feasible to evaluate the functions for all integers between 1 and 10000 with this approach.

Algorithm	`is_sturdy`	swm	msw	mfw
`aut`	31950	31990	42229	41747
`order_deg_bfs`	7378	164543	439794	—
`bfs01`	5209	5207	15515	41761

10 Computational results

Sequence A143027 in the OEIS [27] gives a list of the first few sturdy primes, namely,

$$2, 3, 5, 7, 17, 31, 73, 89, 127, 257, 1801, 2089, 8191, 65537, 131071,$$
$$178481, 262657, 524287, 2099863,$$

and mentions 616318177 as an additional sturdy prime, although it was not known if this was the next sturdy prime to occur in the sequence. Using our methods, we checked all primes $p < 2^{32}$. We confirmed the results in the OEIS and found that 616318177 and 2147483647 are the only remaining sturdy primes in that range.

We also computed frequency counts for the values of swm for odd $n > 1$, not just primes, and they are given in Table 4.

11 Numbers with few 0's

We can also use finite automata to determine when numbers with few 0's are flimsy. More precisely, for each pair of integers j, k we can build a DFA $M_2(j, k)$ accepting those $(n)_2$ for which $(n)_2$ has j 0's and $(kn)_2$ has more than $j + t$ 0's, where $t = |(kn)_2| - |(n)_2|$. Such an n is guaranteed to be flimsy. We can determine t by reading the input n, least significant digit first, and computing $(kn)_2$ on the fly, keeping track of the carries.

Let j be a fixed natural number. By choosing an appropriate set of flimsy witnesses k (which can be guessed empirically), we can determine all flimsy numbers having exactly j 0's in their binary representation. We do this by computing the DFA's $M_2(j, k)$ and unioning them together to get a final automaton M'_j. We expect there to be a finite set of "sporadic" sturdy exceptions, and (according to Theorem 11) an infinite set of sturdy exceptions consisting of those numbers with binary representation of the form $s1^i\overline{s}$, where s begins with 1 and ends with 0. This expectation can then be verified by considering the language accepted by M'_j; the finite set of sturdy exceptions can be tested using our algorithms previously discussed. The multipliers we used in constructing M'_j are the odd numbers $\leq 2^{j+1} + 1$.

Table 4 Counts of swm(n) for odd $n > 1$.

swm(n)	$n < 2^{20}$	$2^{20} < n < 2^{21}$	$2^{21} < n < 2^{22}$	$2^{22} < n < 2^{23}$
2	115931	107650	208333	403823
3	286681	294938	596522	1205753
4	83895	83958	168138	336448
5	19287	19242	38566	77071
6	9903	9892	19812	39635
7	4246	4265	8510	17023
8	2274	2269	4548	9104
9	1027	1030	2058	4119
10	529	527	1059	2118
11	256	257	514	1024
12	130	131	260	521
13	64	64	128	256
14	32	33	64	129
15	16	16	32	64
16	8	8	16	32
17	4	4	8	16
18	2	2	4	8
19	1	1	2	4
20	1	0	1	2
21	0	1	0	1
22	0	0	1	0
23	0	0	0	1

We also computed counts of odd sturdy numbers up to 10^i for $i = 1, 2, 3, 4, 5, 6$, and they are given below:

Table 5 Counts of sturdy numbers.

i	Number of odd sturdy numbers $< 10^i$
1	5
2	22
3	81
4	292
5	995
6	3438

With these ideas we can prove the following theorem.

▶ **Theorem 16.**

(a) *Every integer with no* 0*'s is sturdy.*

(b) *Every odd integer with one* 0 *is flimsy, with the exception of* $5 = [101]_2$*, and is proven flimsy by multiplier* 3 *or* 5*.*

(c) *Every odd integer with two* 0*'s is flimsy, with the exception of* 51 *and numbers of the form* 101^i01*,* $i \geq 0$*, which are all sturdy.*

(d) *Every odd integer with three* 0*'s is flimsy, with the exception of* $17, 85, 89, 455$ *and numbers of the form* 1001^i011 *or* 1101^i001*,* $i \geq 0$*, which are all sturdy.*

(e) *Every odd integer with four* 0*'s is flimsy, with the exception of* 33, 69, 73, 153, 3855*, and numbers of the form* 10001^i0111*,* 11001^i0011*,* 10101^i0101*,* 11101^i0001*,* $i \geq 0$*, which are all sturdy.*

(f) *Every odd integer with five* 0*'s is flimsy, with the exception of* 65, 133, 161, 267, 275, 1365, 31775*, and numbers specified by Theorem 11.*

(g) *Every odd integer with six* 0*'s is flimsy, with the exception of* 129, 259, 261, 273, 385, 525, 549, 561, 585, 645, 657, 705, 771, 777, 801, 1729, 1801, 2275, 3185, 11565, 13107, 258111*, and numbers specified by Theorem 11.*

(h) *Every odd integer with seven* 0*'s is flimsy, with the exception of* 257, 515, 517, 529, 1035, 1065, 1105, 1155, 1157, 1185, 1285, 1545, 1665, 2077, 2201, 2325, 2449, 2573, 2697, 2821, 2945, 19065, 19275, 21845, 26985, 95325, 2080895*, and numbers specified by Theorem 11.*

(i) *Every odd integer with eight* 0*'s is flimsy, with the exception of* 513, 1027, 1029, 1057, 1281, 2055, 2085, 2089, 2097, 2115, 2145, 2193, 2313, 2337, 2563, 2565, 2625, 3075, 3105, 3585, 4123, 4185, 4371, 4389, 4433, 4619, 4675, 4681, 4867, 4929, 5187, 6169, 6417, 6665, 6913, 8253, 8505, 8525, 8645, 8757, 9009, 9261, 9513, 9765, 10017, 10269, 10465, 10521, 10773, 11025, 11277, 11529, 11781, 12033, 12483, 13505, 14497, 18631, 25623, 34695, 39321, 42405, 50115, 57825, 158875, 222425, 774333, 16711935*, and numbers specified by Theorem 11.*

(j) *Every odd integer with nine* 0*'s is flimsy, with the exception of* 1025, 2051, 2057, 2065, 2177, 3073, 4131, 4165, 4233, 4361, 4369, 4417, 4641, 5129, 5185, 6273, 8215, 8277, 8339, 8401, 8711, 8773, 8835, 8897, 10261, 10385, 10757, 10881, 12307, 12369, 12803, 12865, 14353, 14849, 16443, 16569, 16835, 16947, 17073, 17451, 17577, 17745, 17955, 18081, 18459, 18585, 18963, 19089, 19467, 19593, 19971, 20097, 24605, 24633, 25025, 25137, 25641, 26145, 26649, 26691, 27153, 27657, 28161, 28679, 32893, 33401, 33909, 34417, 34925, 35433, 35941, 36449, 36957, 37465, 37973, 38481, 38989, 39497, 40005, 40513, 41021, 41529, 41769, 42037, 42545, 43053, 43561, 44069, 44577, 45085, 45593, 46101, 46609, 47117, 47625, 48133, 48641, 178481, 285975, 349525, 413075, 476625, 1290555, 1806777, 1864135, 6242685, 133956095*, and numbers specified by Theorem 11.*

Based on this theorem, we make the following conjecture.

▶ **Conjecture 17.** Every number with j 0's is flimsy, with exceptions of the form $s1^i\overline{s}$, $i \geq 0$, where $|s| = j$ and s begins with 1 and ends with 0, and only finitely many additional exceptions.

12 The k-flimsy numbers via formal language theory

In this section we describe a new approach, based on formal language theory, for understanding the distribution of the k-flimsy numbers. Recall these are the numbers

$$F_k = \{n \geq 1 \ : \ s_2(kn) < s_2(n)\}.$$

The majority of the results in this section are about the case $k = 3$, although in principle our technique can be applied to any odd k.

Kátai [16] studied the difference $s_2(3n) - s_2(n)$, and proved that this quantity is essentially normally distributed, in a certain sense. Stolarsky [28] conjectured that the natural density of the k-flimsy numbers is 1/2 for all odd k. His conjecture was later proved by W. M. Schmidt [25] and J. Schmid [24]. All these results use rather sophisticated tools of number theory and probability.

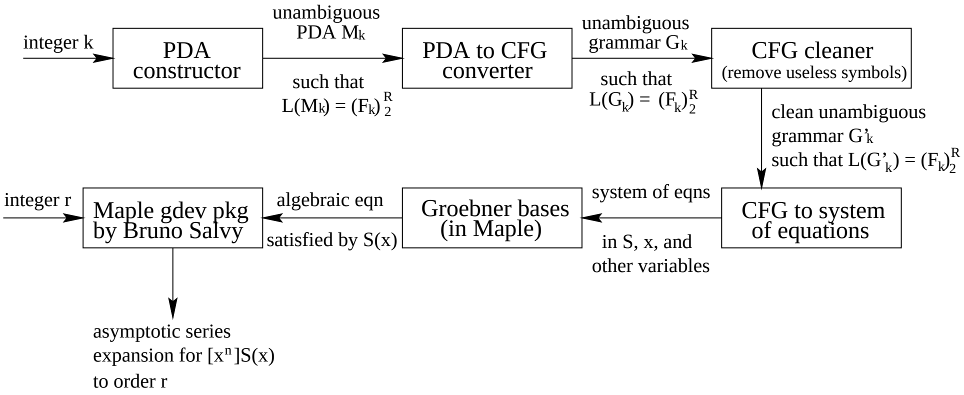
In contrast, in this section we obtain rather detailed results on the distribution of 3-flimsy numbers through a (more or less) purely mechanical approach based on formal language theory. The main result of this section is the following:

▶ **Theorem 18.** *The number of* 3*-flimsy numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left(\frac{1}{4} - cN^{-1/2} + O(N^{-3/2}) \right), \tag{1}$$

where $c = \frac{7\sqrt{6}}{24\sqrt{\pi}} \doteq 0.4030765$.

Our method starts with a pushdown automaton (PDA) recognizing the k-flimsy numbers, and by a series of steps, it is converted into an asymptotic series expansion for the number of k-flimsy numbers with N bits. Previously, the basic approach has been used for a wide variety of combinatorial enumerations; see, for example, [4, 5, 2, 3]. We have implemented all the steps, and the flow of control is explained in the diagram below.


integer k
PDA constructor
unambiguous PDA M_k
such that L(M_k) = (F_k)^R_2
PDA to CFG converter
unambiguous grammar G_k
such that L(G_k) = (F_k)^R_2
CFG cleaner (remove useless symbols)
clean unambiguous grammar G'_k such that L(G'_k) = (F_k)^R_2
CFG to system of equations
system of eqns
in S, x, and other variables
Groebner bases (in Maple)
algebraic eqn
satisfied by S(x)
integer r
Maple gdev pkg by Bruno Salvy
asymptotic series expansion for [x^n]S(x) to order r

We now explain briefly what each box in the diagram does, with more detailed explanation to follow. For all undefined terms, see any textbook on automata theory or formal languages, such as [15].

First, given an odd integer $k \geq 3$, we build an unambiguous pushdown automaton (PDA) M_k that recognizes the base-2 representation of elements of F_k; more precisely, M_k recognizes the language $(F_k)_2^R$. The length-N strings in $(F_k)_2^R$ are in 1-1 correspondence with the flimsy numbers in the half-open interval $[2^{N-1}, 2^N)$, so our goal is to estimate the cardinality of $(F_k)_2^R \cap \{0,1\}^N$ as precisely as possible.

Second, we convert M_k to an unambiguous context-free grammar G_k generating $(F_k)_2^R$.

We simplify this context-free grammar by deleting useless symbols (those symbols that do not participate in the derivation of any terminal string, or are not reachable from the start variable), obtaining a new CFG G'_k.

Third, we convert G'_k to a system of equations in the variables of G'_k. These variables represent formal power series, with the property that the number of length-N strings generated by a variable A is given by $[x^N]A(x)$, the coefficient of x^N in the power series $A(x)$.

Fourth, using Gröbner bases, we solve this system of equations, obtaining an algebraic equation satisfied by the formal power series $S(x)$, where S is the start variable of the grammar G'_k.

Finally, using Bruno Salvy's `gdev` package, written in Maple, we can determine the asymptotic behavior of $[x^N]S(x)$ using the saddle-point method (as discussed by, e.g., Flajolet and Sedgewick [12]). In principle, we can obtain as many terms as we wish of the asymptotic expansion.

Theorem 18 now follows by performing each of these steps. The first four steps are done with original code written by the first author in Python, and the last two steps are done with Maple. The code for each step is available at

`https://git.uwaterloo.ca/Flimsy/CFLpy`.

We now give more complete details of some of the steps.

12.1 Constructing the PDA M_k

The general idea is as follows: we create a PDA accepting the base-2 representation of k-flimsy numbers n. We use the stack of the PDA to record the absolute value of $s_2(n) - s_2(kn)$, and we use the state to record both the carry needed when multiplying input by k, and the sign of $s_2(n) - s_2(kn)$. We accept the input if the carry is 0, the sign of $s_2(n) - s_2(kn)$ is positive, and the stack has at least one counter.

Our PDA is assumed to begin its computation with a special symbol, `Z`, on top of the stack, and if the input is accepted, to end its computation when the stack becomes empty.

The sketch above is not quite enough because of two technical issues. First, (a) in some cases this approach requires reading extra leading zeroes (which, because we are representing numbers starting with the least significant digit first, would be at the end of the input), in order to guarantee that the carry for $s_2(kn)$ was taken into account and (b) we must have that the leading bit of the input is 1, to avoid incorrectly counting smaller numbers as having n bits.

To handle both these issues, we slightly modify the construction in several ways. First, if the state has a minus sign, then the stack holds $|y|_1 - |x|_1$ `X`'s, where x is the input seen so far and y is the $|x|$ least significant bits of $k(x)_2^R$. On the other hand, if the state has a positive sign, then the stack holds $|x|_1 - |y|_1 - 1$ `X`'s.

Second, to simulate the needed leading zeroes required to handle the carry, without actually reading them, we use a special series of $\log_2 k$ states to pop `X`'s from the stack.

Finally, we have a special state used to empty the stack when acceptance is detected. The total number of states is therefore at most $2k + \log_2 k$. The resulting PDA M_3 is depicted in Figure 1.

One important property of our construction is that our PDA M_k is *unambiguous*. By this we mean that every accepted word has exactly one accepting computational path.

12.2 Converting the PDA to a CFG

We can convert M_k to an equivalent context-free grammar G_k using a standard technique called the "triple construction" [15, pp. 115–119]. This gives us a grammar G_k with $O(k^2)$ variables and $O(k^3)$ productions.

Now we use the fact, proved in [13, Thm. 5.4.3, p. 151], that performing the triple construction on an unambiguous PDA gives us an unambiguous grammar.

12.3 Cleaning the CFG

We can remove useless symbols from our grammar G_k by removing all variables that do not derive a terminal string, then removing all productions containing these removed variables, and then removing all variables and terminals that are not reachable from the start variable. This is a standard procedure, and is described in greater detail in [15, pp. 88–90].

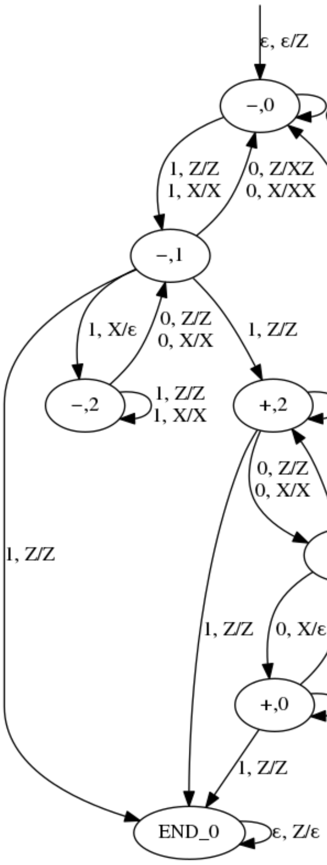


Figure 1 PDA M_3.

Once this is complete, it may be found that there is a variable X that has only one production $X \to \alpha$. If X is not the start variable, then it can be deleted from the set of variables, and all instances of X in production rules can be replaced with α.

For example, when we convert our PDA M_3, we get an unambiguous grammar G_3; cleaning G_3 using this procedure gives us the following grammar G_3':

$$\begin{array}{ll}
S \to 1F \mid 0S & A \to 1E \mid 0A \\
B \to 1G \mid 0B & C \to 1H \mid 1 \mid 0C \\
D \to 1I \mid 0D & E \to 1 \mid 0AJ \\
F \to 1N \mid 0AK & G \to 1LB \mid 0 \\
H \to 1M \mid 1LC \mid 1 & I \to 1M \mid 1LD \mid 1 \mid 0S \\
J \to 1J \mid 0E & K \to 1K \mid 0F \\
L \to 1L \mid 0G & M \to 1M \mid 1 \mid 0H \\
N \to 1N \mid 0I &
\end{array}$$

12.4 Converting the CFG to a system of equations

This transformation was discussed in [9]. It suffices to replace, in each set of productions $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_i$ of a grammar G, each terminal symbol by the indeterminate x, each $\mid$ symbol by a plus sign, and the $\to$ with an equals sign. For a proof of correctness, see [17, 19].

Performing this transformation on G'_3 gives us the following system of equations:

$$\begin{aligned} S &= xF + xS & A &= xE + xA \\ B &= xG + xB & C &= xH + x + xC \\ D &= xI + xD & E &= x + xAJ \\ F &= xN + xAK & G &= xLB + x \\ H &= xM + xLC + x & I &= xM + xLD + x + xS \\ J &= xJ + xE & K &= xK + xF \\ L &= xL + xG & M &= xM + x + xH \\ N &= xN + xI \end{aligned}$$

12.5 Solving the system

We can now solve the resulting system of equations for S, obtaining an algebraic equation for which S is the root. The main tool is Groebner bases, for which a helpful package already exists in `Maple`.

Using the `Maple` code below, we find the following quadratic equation for S in the case $k = 3$.

$$x(2x-1)^2(x+1)(2x^2-x+1)S(x)^2+(2x-1)(x-1)^2(x+1)(2x^2-x+1)S(x)+x^4(x^2-x+1) = 0.$$

To use this code, you will first need to download the `algolib` package from `http://algo.inria.fr/libraries/`.

```
eqs := [-S + x*V_F + x*S,
-V_A + x*V_E + x*V_A,
-V_B + x*V_G + x*V_B,
-V_C + x*V_H + x + x*V_C,
-V_D + x*V_I + x*V_D,
-V_E + x + x*V_A*V_J,
-V_F + x*V_N + x*V_A*V_K,
-V_G + x*V_L*V_B + x,
-V_H + x*V_M + x*V_L*V_C + x,
-V_I + x*V_M + x*V_L*V_D + x + x*S,
-V_J + x*V_J + x*V_E,
-V_K + x*V_K + x*V_F,
-V_L + x*V_L + x*V_G,
-V_M + x*V_M + x + x*V_H,
-V_N + x*V_N + x*V_I]:
Groebner[Basis](eqs, lexdeg([V_A, V_B, V_C, V_D, V_E, V_F, V_G, V_H,
V_I, V_J, V_K, V_L, V_M, V_N], [S]));
algeq := %[1]:
map(series, [solve(algeq, S)], x);
f := solve(algeq,S);
```

```
ps := f[1]:
assume(x, positive):
series(ps, x, 40);
libname:="<insert current directory path>",libname:
combine(equivalent(ps,x,n,5));
```

Solving this quadratic for S gives

$$S(x) = \frac{-(x-1)^2(x+1)(2x^2-x+1) + \sqrt{-(x-1)(2x-1)(2x^2-x+1)(x^3+x^2-x+1)^2}}{2x(2x-1)(x+1)(2x^2-x+1)}.$$

Since the grammar G_3' is unambiguous, the formal power series $S(x)$ is the census generating function for the set $(F_3)_2^R$. In particular, this means that $[x^N]S(x) = |F_3 \cap [2^{N-1}, 2^N)|$, or in other words, the coefficient of x^N in $S(x)$ is the number k-flimsy numbers in $[2^{N-1}, 2^N)$.

12.6 Asymptotic expansion of the coefficients of the power series

Finally, we use Flajolet-Sedgewick-style asymptotic analysis [12, §VII. 7.1] to determine an asymptotic formula for the N'th coefficient of the power series expansion for $S(x)$. Conveniently, there is a Maple package algolib, written by Bruno Salvy [23], to accomplish this. When we run this on our formula for $S(x)$, we get our desired result.

This completes our discussion of the proof of Theorem 18.

▶ Remark 19. We could easily determine more terms in the asymptotic expansion, if we wanted, using the same ideas. For example, we can find that the number of 3-flimsy numbers in the interval $[2^{N-1}, 2^N)$ is

$$2^N \left(\frac{1}{4} - \frac{\sqrt{6}}{\sqrt{\pi}} \left(\frac{7}{24} N^{-1/2} + \frac{13}{72} N^{-3/2} - \frac{17}{64} N^{-5/2} + \frac{3365}{13824} N^{-7/2} + \cdots \right) \right).$$

▶ **Corollary 20.** *The number of* 3*-flimsy numbers* $< 2^N$ *is* $2^{N-1} - O(2^N N^{-1/2})$.

Proof. For any real number $a > 0$ we have

$$\begin{aligned} 2^N N^{-a} \leq \sum_{1 \leq n \leq N} 2^n n^{-a} &\leq \sum_{1 \leq n \leq N/2} 2^n n^{-a} + \sum_{N/2 < n \leq N} 2^n n^{-a} \\ &\leq \sum_{1 \leq n \leq N/2} 2^n + (N/2)^{-a} \sum_{N/2 < n \leq N} 2^n \\ &\leq 2^{N/2+1} + (N/2)^{-a} 2^{N+1}. \end{aligned}$$

Summing (1) and applying the inequalities above gives the desired result. ◀

▶ **Theorem 21.** *The number of 5-flimsy numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left(\frac{1}{4} - cN^{-1/2} + O(N^{-3/2}) \right), \tag{2}$$

where $c = \frac{3\sqrt{5}}{8\sqrt{\pi}} \doteq 0.473087348$.

Proof. This is determined using the same method as the proof for Theorem 18. The details will appear in the full paper. ◀

We can also use the same ideas to compute the distribution of flimsy numbers in other bases. As an example we proved

▶ **Theorem 22.** *The number of integers in the range* $[3^{N-1}, 3^N)$ *that are* 2*-flimsy in base* 3 *is*

$$3^N\left(\frac{1}{3}+\frac{\sqrt{3}}{\sqrt{\pi}}\left(-\frac{1}{3}N^{-1/2}+\frac{1}{48}N^{-3/2}-\frac{13}{1536}N^{-5/2}-\frac{65}{24576}N^{-7/2}+O(N^{-9/2})\right)\right).$$

Proof. As before. We omit the details. ◀

13 The k-equal numbers via formal language theory

Another quantity of interest is the number of n for which $s_2(n) = s_2(kn)$. We call such n *k-equal.* By generalizing the approach used in Section 12, we can compute how many integers $n \in [2^{N-1}, 2^N)$ are k-equal.

In particular, we modify PDA M_k by changing the transitions to the END states. Whereas M_k transitions to END when reading a 1 if following that 1 with sufficiently many zeros would reach the state $(+, 0)$, instead we want such an input to reach the state $(-, 0)$ with no counters on the stack. With this approach we can prove

▶ **Theorem 23.** *The number of* 3*-equal numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N\left(cN^{-1/2}+O(N^{-3/2})\right), \tag{3}$$

where $c = \frac{\sqrt{6}}{4\sqrt{\pi}} \doteq 0.345494149$.

▶ **Theorem 24.** *The number of* 5*-equal numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N\left(cN^{-1/2}+O(N^{-3/2})\right), \tag{4}$$

where $c = \frac{\sqrt{5}}{4\sqrt{\pi}} \doteq 0.315391565$.

The details will appear in the final paper.

14 Conclusions and open problems

We have shown that techniques from automata theory can be used to solve problems in number theory. For other fun along these lines, see [7, 22].

It would be interesting to understand the distribution of values of $\mathrm{msw}(n)$ and $\mathrm{mfw}(n)$ for n flimsy. We leave this as an open problem.

References

1 B. Alexeev. Minimal DFAs for testing divisibility. *J. Comput. System Sci.*, 69:235–243, 2004.

2 A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns: enumerative aspects. In S. T. Klein et al., editors, *LATA 2018*, volume 10792 of *Lecture Notes in Computer Science*, pages 195–206. Springer-Verlag, 2018.

3 A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns, the vectorial kernel method, and generating functions for pushdown automata. *Algorithmica*, 82:386–428, 2020.

4 C. Banderier and M. Drmota. Coefficients of algebraic functions: formulae and asymptotics. In *FPSAC 2013*, volume AS of *DMTCS Proc.*, pages 1065–1076. DMTCS, 2013.

5 C. Banderier and M. Drmota. Formulae and asymptotics for coefficients of algebraic functions. *Combin. Prob. Comput.*, 24:1–53, 2015.

6 B. Bašić. The existence of n-flimsy numbers in a given base. *Ramanujan J.*, 43:359–369, 2017.

7 J. Bell, K. Hare, and J. Shallit. When is an automatic set an additive basis? *Proc. Amer. Math. Soc. Ser. B*, 5:50–63, 2018.

8 L. H. Y. Chen, H.-K. Hwang, and V. Zacharovas. Distribution of the sum-of-digits function of random integers: a survey. *Prob. Surveys*, 11:177–236, 2014.

9 N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North Holland, Amsterdam, 1963.

10 C. Dartyge, F. Luca, and P. Stănică. On digit sums of multiples of an integer. *J. Number Theory*, 129:2820–2830, 2009.

11 C. Elsholtz. Almost all primes have a multiple of small Hamming weight. *Bull. Austral. Math. Soc.*, 94:224–235, 2016.

12 P. Flajolet and R. Sedgewick. *Analytic Combinatorics.* Cambridge University Press, 2009.

13 M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

14 H. Hasse. Über die Dichte der Primzahlen p, für die einevorgegebene ganz rationale Zahl $a \neq 0$ von gerader bzw. ungerader Ordnung mod p ist. *Math. Annalen*, 166:19–23, 1966.

15 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

16 I. Kátai. Change of the sum of digits by multiplication. *Acta Sci. Math. (Szeged)*, 39:319–328, 1977.

17 W. Kuich and A. Salomaa. *Semirings, Automata, Languages.* Springer-Verlag, 1986.

18 R. W. K. Odoni. A conjecture of Krishnamurthy on decimal periods and some allied problems. *J. Number Theory*, 13:303–319, 1981.

19 A. Panholzer. Gröbner bases and the defining polynomial of a context-free grammar generating function. *J. Automata, Languages, and Combinatorics*, 10:79–97, 2005.

20 Pavel V. Phedotov. Sum of digits of a multiple of a given number (in Russian), 2002. Available at `http://digitsum.narod.ru/Index.htm`.

21 V. Pless, P. Solé, and Z. Qian. Cyclic self-dual Z_4-codes. *Finite Fields Appl.*, 3:48–69, 1997.

22 A. Rajasekaran, J. Shallit, and T. Smith. Additive number theory via automata theory. *Theor. Comput. Sys.*, 64:542–567, 2020.

23 B. Salvy. gdev package of algolib version 17.0. Available at `http://algo.inria.fr/libraries/`, 2013.

24 J. Schmid. The joint distribution of the binary digits of integer multiples. *Acta Arith.*, 43:391–415, 1984.

25 W. M. Schmidt. The joint distributions of the digits of certain integer s-tuples. In P. Erdős, editor, *Studies in Pure Mathematics to the Memory of Paul Turán*, pages 605–622. Birkhäuser, 1983.

26 D. Shanks. Class number, a theory of factorization and genera. In *Proc. Sympos. Pure Math.*, volume 20, pages 415–440, 1969.

27 N. J. A. Sloane et al. The on-line encyclopedia of integer sequences. Available at `https://oeis.org`, 2019.

28 K. B. Stolarsky. Integers whose multiples have anomalous digital frequencies. *Acta Arith.*, 38:117–128, 1980/81.

29 S. S. Wagstaff et al. The Cunningham project. Available at `https://homes.cerias.purdue.edu/~ssw/cun/index.html`, 2019.

Efficient Algorithms for Battleship

Loïc Crombez
Université Clermont Auvergne, LIMOS, Aubière, France
https://fc.isima.fr/~lcrombez/
loic.crombez@uca.fr

Guilherme D. da Fonseca
Université Aix Marseille, LIS, France
https://pageperso.lis-lab.fr/guilherme.fonseca/
guilherme.fonseca@lis-lab.fr

Yan Gerard
Université Clermont Auvergne, LIMOS, Aubière, France
https://yangerard.wordpress.com/
yan.gerard@uca.fr

Abstract

We consider an algorithmic problem inspired by the Battleship game. In the variant of the problem that we investigate, there is a unique ship of shape $S \subset \mathbb{Z}^2$ which has been translated in the lattice $\mathbb{Z}^2$. We assume that a player has already hit the ship with a first shot and the goal is to sink the ship using as few shots as possible, that is, by minimizing the number of missed shots. While the player knows the shape S, which position of S has been hit is not known.

Given a shape S of n lattice points, the minimum number of misses that can be achieved in the worst case by any algorithm is called the Battleship complexity of the shape S and denoted $c(S)$. We prove three bounds on $c(S)$, each considering a different class of shapes. First, we have $c(S) \leq n - 1$ for arbitrary shapes and the bound is tight for parallelogram-free shapes. Second, we provide an algorithm that shows that $c(S) = O(\log n)$ if S is an HV-convex polyomino. Third, we provide an algorithm that shows that $c(S) = O(\log \log n)$ if S is a digital convex set. This last result is obtained through a novel discrete version of the Blaschke-Lebesgue inequality relating the area and the width of any convex body.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Polyomino, digital geometry, decision tree, lattice, HV-convexity, convexity

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.11

Related Version https://arxiv.org/abs/2004.07354

Funding *Loïc Crombez*: This work has been sponsored by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).
Guilherme D. da Fonseca: This work is supported by the French ANR PRC grant ADDS (ANR-19-CE48-0005) and the Brazilian CAPES-PrInt project number 88881.310248/2018-01.
Yan Gerard: This work is supported by the French ANR PRC grant ADDS (ANR-19-CE48-0005).

1 Introduction

We consider a geometric problem inspired by the children's game Battleship. The Wikipedia description of the game is:

> Battleship (also Battleships or Sea Battle) is a strategy type guessing game for two players. It is played on ruled grids (paper or board) on which each player's fleet of ships (including battleships) are marked. The locations of the fleets are concealed from the other player. Players alternate turns calling "shots" at the other player's ships, and the objective of the game is to destroy the opposing player's fleet.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 11; pp. 11:1–11:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

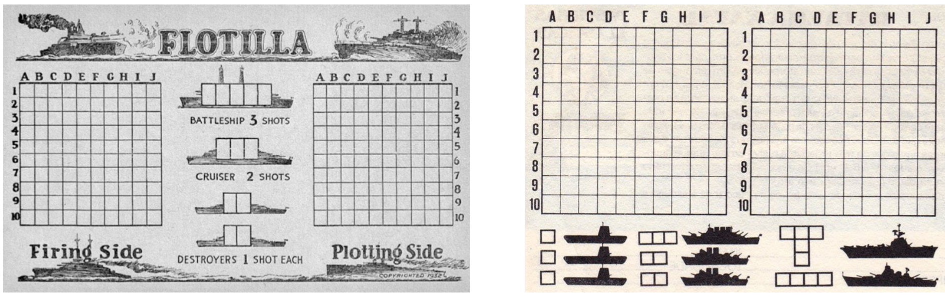


Figure 1 American and Portuguese paper game boards of Battleship.

After each shot, the player is informed if the shot has been a "hit" or a "miss", but no other information is given. In the original version the shapes of the ships are line segments of different lengths. However different countries and commercial brands use a variety of shapes for the ships (see Figure 1).

During the game, the strategy of a player (that we call Alice), is usually decomposed in an alternate sequence of two steps:

1. Hit a new ship of the opponent (that we call Bob).
2. Sink that ship with a minimal number of misses and go back to the first step.

The first step is a hitting set problem and many interesting variations are possible. In this paper, however, we consider the second step. The goal of the second step is to sink the ship (which has already been hit once) with a minimal number of misses. During a real game, the position of the new ship can be constrained by the positions of the other ships which have been already discovered and the grid boundaries, but we consider a simpler case. The fleet is composed of only one ship placed on an infinite grid using only integer translations. In other words, its shape S is given and can only be translated in the lattice. We know, however the coordinates of one grid cell of the ship.

The second modification to the rules that we make is to forbid rotations. This modification simplifies the problem and since there are at most 4 possible rotations, the original problem can be solved by considering each rotation separately (at the expense of a factor of 4).

As a toy example to motivate the problem, we consider the case in which the shape of the ship is a horizontal line segment of length 4. Alice has already hit Bob's ship, which she knows is a horizontal line segment of length 4. However, Alice is clueless about which square of the ship she has hit. In this case, Alice may progressively shoot to the right of the first hit until she misses a shot. At this point, she knows the precise location of Bob's ship and may finish sinking it without missing any additional shot (if she has not already sunk the ship at the fourth shot). In this case, Alice has a strategy that requires at most 1 missed shot.

We refer to the minimum number of misses that Alice needs to sink a ship of shape $S \subset \mathbb{Z}^2$ as $\mathrm{c}(S)$. Notice that Alice knows the shape of S, but not which square she has initially hit. We just showed that $\mathrm{c}(S) \leq 1$ if S is a horizontal (or vertical) line segment. But what happens if the shape S of the ship is not a line segment?

The goal of this paper is to provide bounds to $\mathrm{c}(S)$ depending on properties of the shape S. We prove the following results for a shape S of n points:

- for arbitrary shapes, $\mathrm{c}(S) \leq n - 1$,
- for parallelogram-free shapes, $\mathrm{c}(S) = n - 1$,
- for HV-convex polyominoes, $\mathrm{c}(S) = O(\log n)$, and
- for digital convex shapes, $\mathrm{c}(S) = O(\log \log n)$.

The remainder of the paper is organized as follows. Section 2 is devoted to formalize our notation and to prove some simple results. In Section 3, we provide an algorithm with $O(\log n)$ misses in the worst case for HV-convex polyominoes. In Section 4, we present an algorithm with $O(\log \log n)$ misses in the worst case for digital convex sets. We conclude the paper with a presentation of several open problems and variations.

2 Preliminaries

In this section, we formalize our notation and prove some simple results. Before going further, let us make the notations precise. The ship's *shape* is the finite lattice set $S \subset \mathbb{Z}^2$ and its number of points is n. The opponent translated the shape by an unknown vector $-p \in \mathbb{Z}^2$ obtaining a *ship* $S - p$. The vector $p \in \mathbb{Z}^2$ is the *position* of the ship. We say that a *shot* x is a *hit* if $x \in S - p$ and a *miss* otherwise. By assuming (without loss of generality) that the first hit happens at the origin $x = (0, 0)$, we know then that the position p of the ship is a point in the shape S (that is, $p \in S$) but we do not know which point. In order to determine the actual value of p, we are allowed to test the membership in S of points of the form $p + x$ and our goal is to determine p using as few failed membership tests (called *misses*) as possible.

Given a shape S, we can model an algorithm to determine the position p by a binary decision tree T. The children of each node correspond to the possible outcomes of the shot: hit or miss. The leaves of the decision tree represent the nodes in which the position p of the ship has been determined (they are not necessarily obtained after a hit). Since each leaf corresponds to a different position $p \in S$ of the ship, it follows that there are exactly n leaves.

The efficiency of the algorithm depends on the number of misses in the path going from the root to a leaf corresponding to position p. This number of misses for a position p using tree T is denoted $\mathrm{m}_T(p)$. We omit the subscript T in $\mathrm{m}_T(S)$ when the decision tree T is clear from the context.

The *complexity of the algorithm* T is the maximum number of misses in a path going from the root to a leaf, that is $\max_{p \in S} m_T(p)$. Note that the complexity is generally not equal to the height of the tree.

Given a shape S, we define the Battleship *complexity* $\mathrm{c}(S)$ as the worst-case complexity considering all the decision trees T that determine the position a ship of shape S:

$$\mathrm{c}(S) = \min_T \max_{p \in S} \mathrm{m}_T(p).$$

Next, we reuse the simple example of a horizontal line segment of length 4, to illustrate our notation and framework. In this case, the shape S is the set of lattice points

$$S = \{(0, 0), (1, 0), (2, 0), (3, 0)\}.$$

An optimal algorithm for this shape has already been presented in the Introduction: after the initial shot at $x = (0, 0)$, we shoot at values $x = (1, 0), (2, 0), \ldots$ until a miss occurs. This algorithm is modeled by the decision tree represented in Figure 2. The number of misses $\mathrm{m}(x)$ of this algorithm is equal to 0 if $p = (0, 0)$ and 1 otherwise, giving a maximum of 1, which proves $\mathrm{c}(S) \leq 1$. It is easy to see that for any shape S with $|S| > 1$, $\mathrm{c}(S) \geq 1$. Hence, the algorithm is optimal.

A decision tree for a more complex shape is presented in Figure 3. At each node of the tree, let P be the corresponding set of possible positions. The set P is represented by gray squares in the figure. The inclusion $p \in S$ is the only information that we have about the

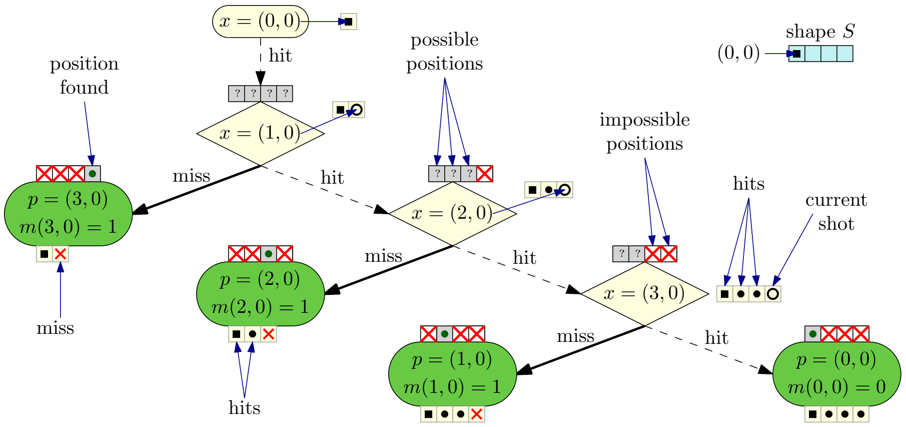


Figure 2 Decision tree modeling an algorithm to sink the ship of horizontal shape $S = \{(0,0),(1,0),(2,0),(3,0)\}$. The nodes correspond to the results of each. At each node, the set P of the possible positions is represented by the gray squares in the small grid. The leaves are the nodes where the ship position has been determined. The worst-case number of misses is 1.

position of the ship when we start the algorithm (at the root of the decision tree). Hence, the set of possible positions at the root is $P = S$ and P gets smaller at each new shot until it is reduced to a singleton at the leaves of the tree. At each new shot, P is reduced in the following way (we use $S - x$ to denote a translation of the set S by vector x):

- If x is a hit, then the set of possible positions for the child becomes $P \leftarrow P \cap (S - x)$.
- If x is a miss, then the set of possible positions for the child becomes $P \leftarrow P \setminus (S - x)$.

It is easy to see that whatever the set of possible positions P is, there always exists a shot which allows us to split P in two non-empty subsets $P \setminus (S - x)$ and $P \cap (S - x)$. Hence, the number of elements in P strictly decreases as we move from a parent to a child and $\mathrm{m}(S) \leq n - 1$ for all shapes S.

2.1 Connection with Classification Trees

Classification trees are decision trees involved for instance in data mining for identifying an element p belonging to a discrete set called the *source set* and denoted S [9]. The element p is identified through the outcomes of a sequence of tests, where the choice of the new test depends on the previous outcomes. This dependency is modeled by a tree whose root represents the initial test. More generally, any internal node is associated to a test T while its children correspond to the possible outcomes of T. Given an unknown element p, the algorithm to identify p starts from the root. At each node, it considers the associated test and goes to the children node corresponding to the outcome of the test. The algorithm stops when arriving at a leaf: the leaf provides the identity of the unknown element p.

The number of tests required to identify p is the level of the corresponding leaf. Then the design of decision trees of small height is a well studied problem. This problem is known to be NP-hard in general [7] (for minimizing the expected level of the leaves). The algorithm that chooses the most balanced test at each node provides an $O(\log n)$-approximation algorithm and the problem admits no polynomial $o(\log(n))$-approximation algorithms [1].

Decision trees have been used in computational geometry for different purposes, for instance determining geometric models [3] or concept classes [2] in an image or more recently for the k-sum problem [8]. As far as we know, the question of designing efficient strategies for playing Battleship with different types of shapes has not been addressed.

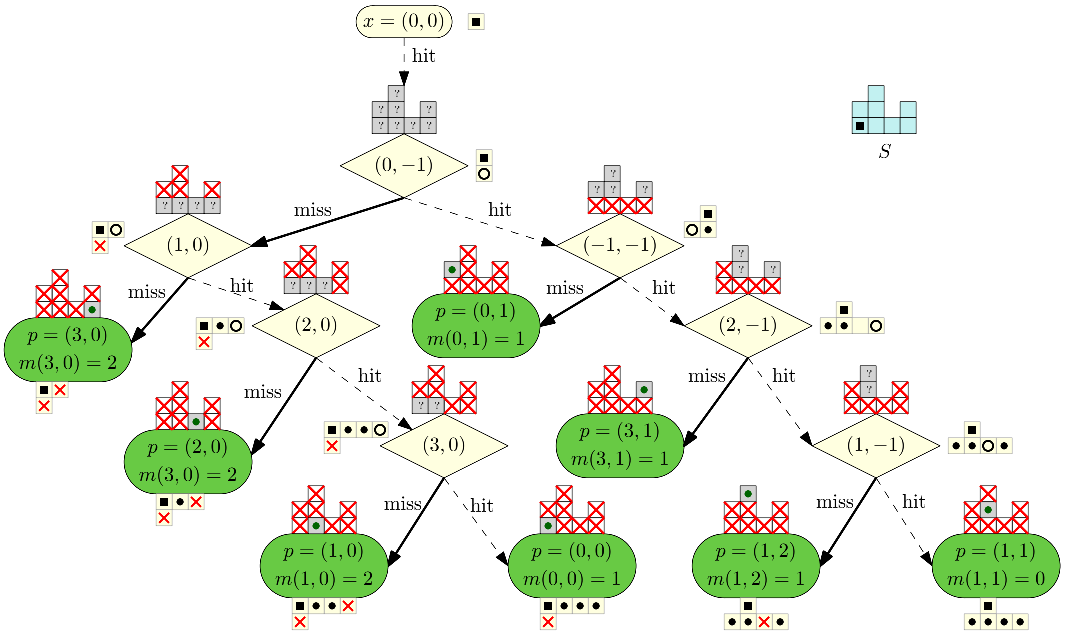


Figure 3 A shape $S \subset \mathbb{Z}^2$ and an algorithm to sink the ship having this shape with at most 2 misses. We follow the same graphic code as in Figure 2. This algorithm T has a worst case complexity $\mathrm{m}(T) = 2$. The worst-case complexity of this shape S is exactly $\mathrm{m}(S) = 2$.

However, our problem possesses a fundamental difference in comparison to the classical use of classification trees: the Battleship problem is not symmetric. The goal in Battleship is to sink the ship with the minimal number of shots, but, since the number of hits to sink a ship is always equal to n, the goal becomes to minimize the number of misses needed to locate the position of the ship.

The simplest heuristic to design classification trees of small height is to choose at each node the most balanced test possible. In our case, that would mean to choose a test such that the number of elements in $P \cap (S - x)$ and $P \setminus (S - x)$ are as similar as possible. As our goal is to minimize the number of misses instead of the height, we believe that a good heuristic strategy is to choose a shot such the number $|P \setminus (S - x))|$ of possible positions in case of a miss is as small as possible. However, we have not been able to prove any good worst-case bounds for this heuristic.

2.2 Parallelogram-Free Shapes

We say that a set S is *parallelogram-free* if every pair of distinct points define a unique difference vector (see Figure 4 for an example). In other words, S does not contain two distinct pairs of distinct points $s_1 \neq s_2$, $s_3 \neq s_4$ such that $s_2 - s_1 = s_4 - s_3$. In this section, we show that if S is parallelogram-free, then $\mathrm{c}(S) = n - 1$.

▶ **Theorem 1.** *If S is a parallelogram-free polyomino of n points, then $\mathrm{c}(S) = n - 1$.*

Proof. We have already showed that $\mathrm{c}(S) \leq n - 1$ for any shape. Next, we show that $\mathrm{c}(S) \geq n - 1$ for a parallelogram-free shape S. To do this, we show that whenever we obtain a hit in the tree, we have successfully determined the position p. Hence, the miss branch of the tree contains all remaining points.

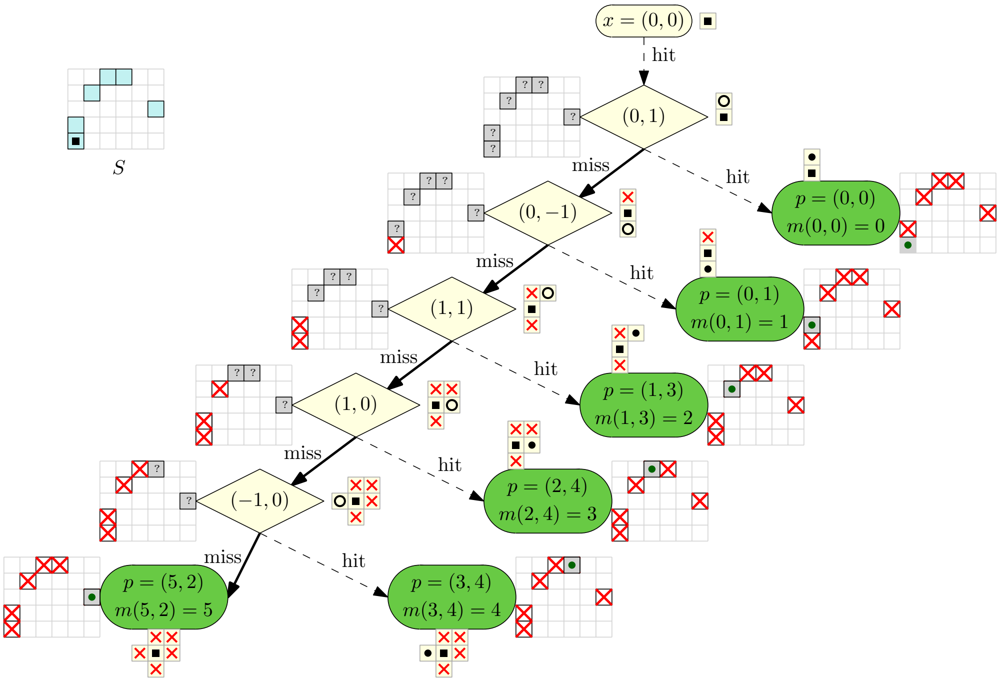


Figure 4 A parallelogram-free shape $S \subset \mathbb{Z}^2$. In this case, all algorithms have isomorphic decision trees.

At any node of the tree, the new set of positions after a hit by shooting $x \neq (0,0)$ is defined by $P \leftarrow P \cap (S - x)$. Let us assume to obtain a contradiction that y and y' are distinct points in $S \cap (S - x)$, then $y = s - x$ and $y' = s' - x$ with again s and s' both in S. It follows that $x = y - s = y' - s'$. As x is not $(0,0)$ and due to the parallelogram-free property, $y = y'$ which contradicts the assumption. ◀

3 HV-Convex Polyominoes

In this Section, we investigate the Battleship complexity for the class of lattice sets of $\mathbb{Z}^2$ which are 4-connected and HV-convex.

3.1 Definition and Properties

A 4-*connected path* is a sequence $(x_1, \ldots, x_k)$ of distinct points of $\mathbb{Z}^2$ such that the Euclidean distance between x_i and x_{i+1} is equal to 1 for $i = 1, \ldots, k-1$. A lattice set $S \subset \mathbb{Z}^2$ is 4-*connected* if for any pair of points $x_1, x_k \in S$, there exists at least one 4-connected path $(x_1, \ldots, x_k)$ in S. The 4-connected finite lattice sets are called *polyominoes* (Figure 5).

HV-convexity is a notion of directional convexity. A lattice set $S \subset \mathbb{Z}^2$ is horizontally (vertically) convex if the intersection of S with any row (column) is a set of consecutive points. A lattice set which is horizontally and vertically convex is said to be *HV-convex* (Figure 5).

We state two properties used in the following for proving the $O(\log n)$ bound on the complexity of HV-convex polyominoes.

▶ **Lemma 2.** *Let S be an HV-convex polyomino. If (x, y) and (x', y') are two different points of S with $x \leq x'$ and $y \leq y'$, then either $(x + 1, y)$ or $(x, y + 1)$ is in S.*

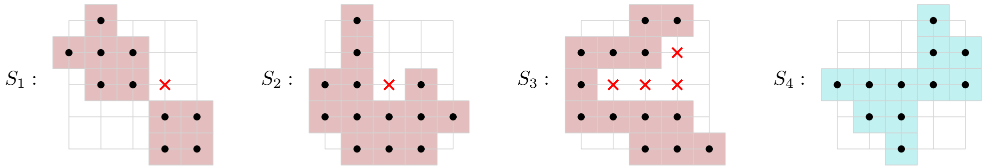


Figure 5 Definition of HV-convex polyominoes. The set S_1 is not 4-connected, hence not a polyomino. The set S_2 a polyomino but not horizontally convex. The set S_3 is a polyomino but not vertically convex. The set S_4 is an HV-convex polyomino since it is 4-connected, horizontally and vertically convex.

Proof. If $x = x'$ or $y = y'$, the result is a direct consequence of the horizontal vertical convexities. We consider now the case $x < x'$ and $y < y'$. Let us assume that neither $(x+1, y)$ nor $(x, y+1)$ are in S, to obtain a contradiction. It follows that the vertical ray above (x, y) and the horizontal ray to the right of (x, y) do not contain any point of S. Then there is no 4-connected path to connect (x, y) and (x', y'). ◀

After this general lemma about HV-convex polyominoes, let us introduce more specific material for our purpose, where we consider only the rows of fixed length ℓ (Figure 6): given an HV-convex polyomino S and a fixed length $\ell \in \mathbb{Z}^+$, let L be the number of rows of length ℓ. For i from 1 to L, we denote by r_i the right endpoints of the i-th row of length ℓ ordered by y coordinate. It follows that for all i, we have (i) $r_i \in S$, (ii) $r_i - (\ell, 0) \notin S$, (iii) $r_i - (\ell - 1, 0) \in S$, and (iv) $r_i + (1, 0) \notin S$.

▶ **Lemma 3.** *Given an HV-convex polyomino S and a fixed length $\ell \in \mathbb{Z}$. The x-coordinate x_i of the right endpoints r_i of the rows of length ℓ forms a monotonic sequence (either $x_i \leq x_{i+1}$ for all $i \in \{1, \dots, L-1\}$ or $x_i \geq x_{i+1}$ for all $i \in \{1, \dots, L-1\}$, as in Figure 6(b)).*

Proof. If the sequence of x-coordinates x_i is not monotonic, then there exists a triplet of indices i, i' and i'' with $i < i' < i''$ leading to a configuration which is neither $x_i \leq x_{i'} \leq x_{i''}$ nor $x_{i''} \leq x_{i'} \leq x_i$. There are 4 remaining permutations that cannot happen in an HV-convex polyomino. We show that it is not possible to have $x_{i'} < x_i \leq x_{i''}$ (see Figure 6(c)), the other 3 cases being analogous. Suppose it is the case in order to reach a contradiction.

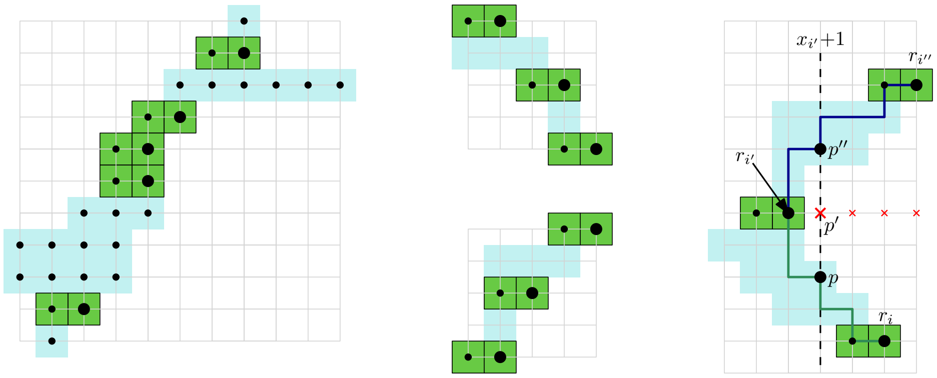
By definition, there is no point in S to the right of $r_{i'}$. Hence every 4-connected path from r_i to $r_{i'}$ intersects the vertical ray going down from $p' = (x_{i'} + 1, 0)$. Let p be a point in this intersection. Similarly, let p'' be a point in the intersection of the path connecting $r_{i''}$ to $r_{i'}$ that is in the ray going up from p'. All p, p', and p'' have the same y coordinate, but $p' \notin S$ while p, p'' are in S, which contradicts vertical convexity. ◀

3.2 Shooting Algorithm with $O(\log n)$ Misses

The previous lemmas allow us to develop an efficient algorithm for shapes that are HV-convex polyominoes.

▶ **Theorem 4.** *For any HV-convex polyomino S of n points, the Battleship complexity* $\mathrm{c}(S) = O(\log n)$.

We notice that the result does not hold for either HV-convex lattice sets (HV-convexity alone does not forbid arbitrarily large parallelogram-free lattice sets), or for arbitrary polyominoes (we let the reader construct counter-examples as a tricky exercise).



(a) (b) (c)

Figure 6 Monotonicity of the right endpoints coordinates for the rows of fixed length ℓ of an HV-convex polyomino (here, $\ell = 2$). (a) Lemma 3 states that for any length, the sequence of the x-coordinates of the right endpoints of the rows of a fixed length ℓ is monotonic (either increasing or decreasing). (b) Compatible configurations. (c) Non-monoticity is not compatible with the HV-convexity of a polyomino.

We prove the $O(\log n)$ bound of Theorem 4 by providing a shooting algorithm with at most $O(\log n)$ misses for locating the position of the ship. We call it the *staircase shooting algorithm.*

The Staircase Shooting Algorithm

The staircase shooting algorithm for HV-convex polyominoes works in two phases. The first phase consists of two sequences of horizontal shots going away from the origin in both horizontal directions. First we shoot at $(k, 0)$ with an increasing $k = 1, \ldots, k^+$ until we obtain a miss at $k = k^+$. We proceed similarly in the negative direction until we obtain a miss at $k = k^-$. This way, we determine two values $k^- < 0$ and $k^+ > 0$ such that all the shots $(k, 0)$ with $k^- < k < k^+$ are hits while $(k^-, 0)$ and $(k^+, 0)$ are both misses. The difference $k^+ - k^- + 1$ provides the length ℓ of the row of S containing the unknown position p. We used 2 misses to obtain the value of ℓ and concluded the first phase.

We now present the decision tree of the remainder of our algorithm. At any node, the set P corresponds to the set of possible positions. The number of rows of length ℓ in S is denoted L. After the first phase, we know that the unknown position p belongs to one of these L rows of length ℓ and we know the horizontal position is the k^--th point of the row. Hence, at this point, we have $|P| = L$ with at most one position in P for each row. The positions are denoted p_i for $1 \leq i \leq |P|$, ordered by y-coordinates. It follows from Lemma 3 that the sequence of the x-coordinates of p_i is either increasing or decreasing. We assume without loss of generality that the sequence is increasing, the other case being analogous. The problem is to further reduce the set P of possible positions. At any point, if $|P|$ is at most 2, then we distinguish the 2 possible positions with only 1 additional miss.

We now describe the second phase, assuming $|P| \geq 3$. The sequence of hits follows a monotone 4-connected path $(s_1, \ldots, s_J)$ of J points of S with $s_0 = (k^+ - 1, 0)$ and shaped as a staircase going up and to the right, that is either $s_{j+1} = s_j + (1, 0)$ or $s_{j+1} = s_j + (0, 1)$. For each j, we have to choose s_{j+1} among the two possibilities. We proceed as in the heuristic

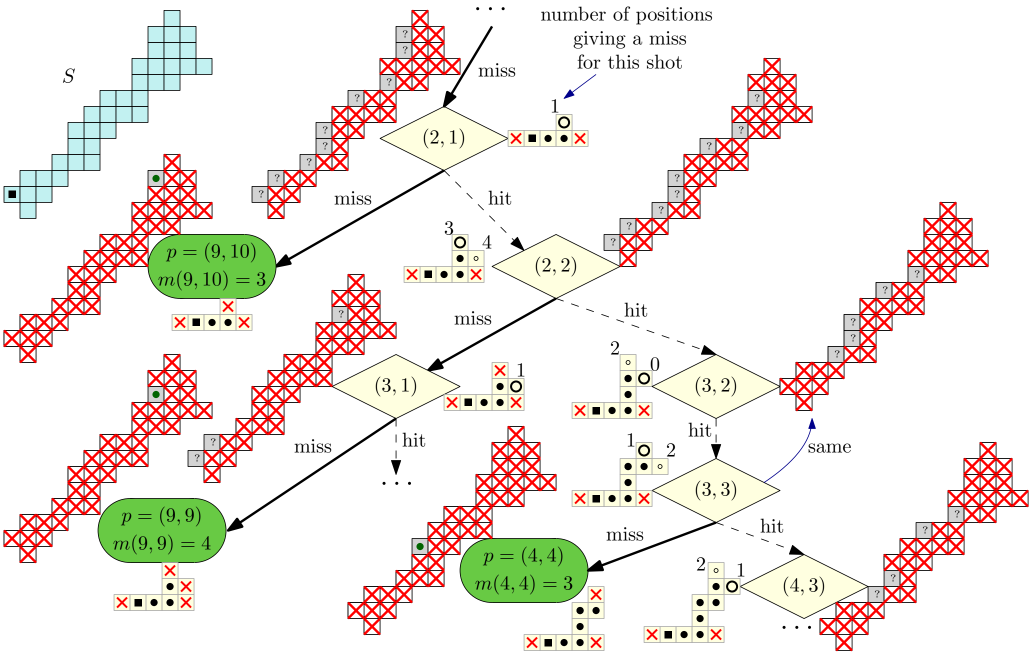


Figure 7 Second phase of the staircase algorithm on an HV-convex polyomino. Only part of the tree is represented.

described in Section 2.1, choosing to shoot at the position x that minimizes the number of positions $p_i \in P$ for which we would have a miss and let x' denote the other choice. We now consider the two possible outcomes after shooting at x.

- The child node after a *hit*: At this new node, we have a new set of positions P (which may or may not have been reduced), an unchanged number of misses, and a new node $s_{j+1} = x$ appended to the path.
- The child node after a *miss*: In this case, the number of misses increased by 1 and we cannot append x to the path, since $x \notin S$. We proceed with another shot at position x' set to the other possibility to build the staircase path. If x' is also a miss, then we determined that the position p is the top-most point of P namely $p = p_{|P|}$ (Claim 5). However, if x' is a hit, then we append $s_{j+1} = x'$ to the staircase path. In this case, we will show that at least 1/3 of possible positions P have been discarded (Claim 6).

Proof of the Claims

We consider the conditions of the algorithm at a current node associated to a monotonous staircase path $(s_1, \dots, s_J)$ with $s_1 = (k^+ - 1, 0)$ such that all shots s_j for $j = 1, \dots, J$ provided hits. The set of the possible positions is a set of $|P|$ points $p_1, \dots, p_{|P|}$ at the k^--th position in rows of length ℓ. The path $s_1, \dots, s_J$ being all hits, we know that for any possible position p_i and any shot s_j of the path, the sum $p_i + s_j$ is a point of S. We remind the reader that the points p_i are ordered by y-coordinates and that we have assumed without loss of generality that their x-coordinates are increasing.

▷ Claim 5. We consider the two new possible shots $s_j + (1, 0)$ or $s_j + (0, 1)$. For all possible positions p_i with $i < |P|$, one of the two shots provides a hit. In other words, the unique possible position for which we might obtain two misses is the top-most $p_{|P|}$.

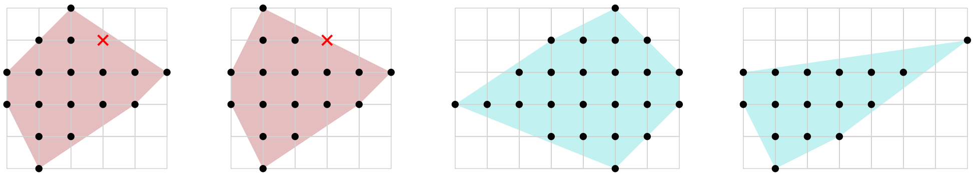


Figure 8 Digital convexity. The two lattice sets on the left are not digital convex while the two on the right are, since there is no other lattice point in their convex hulls.

Proof. We have to prove that for any possible position p_i with $i < |P|$, then either $s_j + (1, 0)$ or $s_j + (0, 1)$ provides a hit. It means that either $p_i + s_j + (1, 0) \in S$ or $p_i + s_j + (0, 1) \in S$. This property is a direct consequence of Lemma 2, because $p_i + s_j \in S$, $p_{|P|} + s_j \in S$, and $p_{|P|} + s_j$ is in the northeast quadrant of $p_i + s_j$ (namely the x and y-coordinates of $p_i + s_j$ are respectively lower than the ones of $p_{|P|} + s_j$). ◁

Claim 5 leads to a second claim under the assumption that $|P| \geq 3$.

▷ Claim 6. At each current node with $|P| \geq 3$, by choosing between $s_j + (1, 0)$ or $s_j + (0, 1)$ the shot for which the number of possible positions $(p_i)_{1 \leq i \leq I}$ remains the largest, the number of positions providing a miss is at most $\frac{2}{3}|P|$.

Proof. Let $P_{(1,0)}$ be the set of positions p_i such that $p_i + s_J + (1, 0) \in S$ and let $P_{(0,1)}$ be the set of of positions p_i such that $p_i + s_J + (0, 1) \in S$. Let $n_{(1,0)} = |P_{(1,0)}|$ and $n_{(0,1)} = |P_{(0,1)}$. According to the claim 5, with the exception of the topmost possible position $p_{|P|}$, all the others give a hit for one of the two possible shots. Hence, we have $n_{(1,0)} + n_{(0,1)} \geq |P| - 1$ and

$$\max(n_{(1,0)}, n_{(0,1)}) \geq \frac{|P| - 1}{2} \geq \frac{|P|}{3},$$

since $|P| \geq 3$. Then with the shot $s_J + (1, 0)$ or $s_J + (0, 1)$ corresponding to the maximum of $n_{(1,0)}, n_{(0,1)}$, we have at least $1/3$ of the possible positions giving a hit. ◁

Complexity Analysis

We need to count the number of misses that the staircase algorithm takes in the worst case. The first phase of the staircase algorithm uses 2 misses. In the second phase, until the number of possible positions falls under 3, each miss reduces the number of possible positions by a factor of at most $2/3$ of the previous value. Since the initial value of $|P|$ is at most n, the number of misses during this phase is at most k where k is the smallest integer verifying $(2/3)^k n \leq 3$. It follows that $k = O(\log n)$. In the last step, we finish the computation with at most 2 more misses.

4 Digital Convex Sets

A shape $S \subset \mathbb{Z}^2$ is *digital convex* if there exists a convex polygon $K \subset \mathbb{R}^2$ such that $S = K \cap \mathbb{Z}^2$. Digital convexity is a stronger property than HV-convexity, however it does not imply 4-connectivity. We can test if a set $S \subset \mathbb{Z}^2$ is digital convex by verifying if $\text{conv}(S) \cap \mathbb{Z}^2 = S$ (Figure 8), where $\text{conv}(S)$ denotes the continuous convex hull of S. This property is exploited in [6] to test digital convexity in linear time.

In this section, we investigate the Battleship complexity of digital convex sets. We choose this family of lattice sets not only because it is one of the main classes of geometric shapes but also because in a continuous variant of the problem, the Battleship complexity of the convex sets is bounded by a constant. Determining if the same holds for the digital version is an intriguing question. In Section 4.1, we prove a new version of the Blaschke-Lebesgue inequality and in Section 4.2, we present an $O(\log \log n)$ algorithm.

4.1 Discrete Blaschke-Lebesgue Inequality

Digital convex sets have inequalities relating the lattice diameter and the lattice width. First, we recall their respective definitions. Let S be a digital convex set. Its *lattice diameter* $d(S)$ is the maximum number of points of S on a line, minus 1 (Figure 9). It follows that the lattice diameter of a single point is 0. A *Diophantine line* is a line containing at least two lattice points. Two Diophantine lines are *consecutive* if they are parallel to each other and there is no lattice point between them. The *lattice width* $w(S)$ of S is the minimum number of consecutive Diophantine lines covering S, minus 1 (Figure 9). The lattice width of a single point is again 0. More formally, the lattice width can be expressed as

$$w(S) = \min_{u \in \mathbb{Z}^2 \setminus \{(0,0)\}} \max_{a,b \in S} u \cdot (b - a).$$

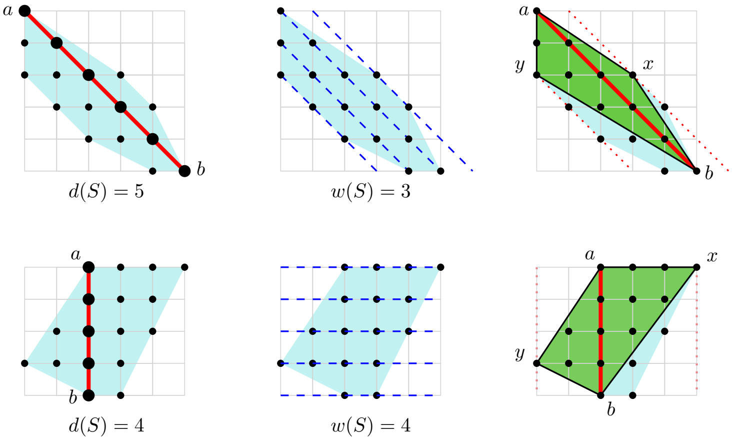


Figure 9 Diameter and lattice width of two lattice sets. On the right, we show the quadrilaterals $xayb$ used in the proof of Lemma 8.

The *continuous width* of a convex body K is the minimum distance between two parallel lines enclosing K and is denoted by $width(K)$. The Blaschke-Lebesgue theorem states that the area of a convex body $K \subset \mathbb{R}^2$ is at least $\frac{\pi - \sqrt{3}}{2} width(K)^2$. This lower bound is achieved when K is the so called Reuleaux triangle. I. Bárány and Z. Füredi [4] provided the following discrete version of the theorem.

▶ **Lemma 7.** *For any digital convex set S, we have $w(S) \leq \lfloor \frac{4}{3} d(S) \rfloor + 1$ and for any fixed diameter, this bound is best possible.*

This inequality is not exactly an equivalent of the Blaschke-Lebesgue theorem since a lower bound on the discrete diameter does not directly provide a lower bound on the area. It remains a small gap to fill in order to obtain a more standard equivalent of the

Blaschke-Lebesgue theorem for digital convex sets. We provide a new discrete inequality closer to the original Blaschke-Lebesgue theorem where the number of points of the lattice set S plays the role of the area and the lattice width plays the role of the width.

▶ **Lemma 8.** *For any digital convex set S of n points, we have $n \geq \frac{3}{8}w(S)^2 - \frac{1}{2}w(S) + 3$.*

Proof. Let us denote a and b the pair of extreme points providing the diameter $d(S)$. It follows $b - a = d(S)v$ where v is the vector in the direction $b - a$ with coprime coordinates. Let u be the rotation of v by $\frac{\pi}{2}$. We consider the points x and y of S minimizing and maximizing the dot product with u. By definition of the lattice width, $u \cdot (y - x) \geq w(S)$ (i). The four points x, a, y, and b define the convex quadrilateral $xayb$ (Figure 9). Its area is $A = \frac{1}{2}|\det(y - x, b - a)| = \frac{1}{2}|d(S)u \cdot (x - y)|$. With (i), we obtain $A \geq \frac{1}{2}d(S)w(S)$ (ii).

Pick's theorem allows us to calculate the number of lattice points in the quadrilateral $xayb$ from its area. We recall the formula $A = i + \frac{e}{2} - 1$ where i is the number of interior points and e the number of points on the boundary of $xayb$. By denoting n_{xayb} the number of lattice points in the quadrilateral $xayb$, we have $n_{xayb} = i + e$. Then Pick's formula provides $n_{xayb} = A + 1 + \frac{e}{2}$. The number of points on the boundary of the quadrilateral being at least 4, we have $n_{xayb} \geq A + 3$. With the bound (ii) on the area A, we obtain $n_{xayb} \geq \frac{1}{2}d(S)w(S) + 3$. As the set S is digital convex, it contains all the lattice points in the quadrilateral $xayb$: $n \geq n_{xayb}$. Then we have $n \geq \frac{1}{2}d(S)w(S) + 3$ (iii).

Lemma 7 provides the bound $w(S) \leq \frac{4}{3}d(S)+1$ which can be rewritten $d(S) \geq \frac{3}{4}(w(S)-1)$. With (iii), it gives $n \geq \frac{3}{8}w(S)^2 - \frac{1}{2}w(S) + 3$. ◀

We can write a similar bound which is easier to use as follows.

▶ **Lemma 9.** *For any digital convex set S of n points, we have $n \geq \frac{1}{4}w(S)^2$.*

Proof. We have $\frac{1}{4}x^2 \leq \frac{3}{8}x^2 - \frac{1}{2}x + 3$ for any real x. Then we can rewrite Lemma 8 with $\frac{1}{4}w(S)^2$ as new lower bound. We could even write $n \geq \frac{1}{4}w(S)^2 + 2$. ◀

Lemma 9 shows that the lattice width is bounded by the square root of the number of points of a digital convex sets. This relation is the key point for proving the complexity of the next algorithm.

4.2 Algorithm with $O(\log \log n)$ Misses

In this section, we prove the following theorem.

▶ **Theorem 10.** *For any digital convex set S of n points, the Battleship complexity $\mathrm{c}(S) = O(\log \log n)$.*

We call this algorithm the *width shooting algorithm* because it is mainly based on shots in the direction given by the lattice width of the set of possible positions. Next, we describe the algorithm.

Consider a node in the tree associated with a set P of possible positions. Initially $P = S$, but as the algorithm progresses, positions will be removed from P. We compute the lattice width $w(P)$, which is achieved by a vector $v \in \mathbb{Z}^2$ with coprime coordinates. Then, we rotate v by $\frac{\pi}{2}$, obtaining a vector u that is parallel to $w(P) + 1$ lines covering the set P. Then, we shoot in directions u and $-u$ with shots of the form ku for positive and negative integer k until we obtain misses at points k^+u and k^-u with hits in between. These two misses and the previous hits lead to a new current node with a new set of possible positions that we denote P'.

We repeat this procedure from node to node until we obtain a set of possible positions whose convex hull has fewer than 25 points. When we reach this value, then any shooting algorithm can be used with at most 24 misses. Theorem 10 follows from the claim that this algorithm has $O(\log\log n)$ misses.

Convex Hull of the Possible Positions

The main procedure of the width shooting algorithm uses shots ku with $k \in \mathbb{Z}$ in direction u until finding the two boundary points of the ship in this direction. After this sequence of shots, due to the digital convexity of S, the new set of possible positions P' has the following property.

▷ Claim 11. The convex hull of P' contains at most one point on each Diophantine line parallel to u.

Proof. Let $k^+ \in \mathbb{Z}^+$ and $k^- \in \mathbb{Z}^-$ be the first positive and negative integers for which the shots ku give a miss (starting from $k = 0$). The difference $k^+ - k^- - 1$ is equal to the length ℓ of the intersection of the shape S and the Diophantine line $p + ku$ for $k \in \mathbb{Z}$. After the two misses obtained with the shots k^+u and k^-u, the set P' of the possible positions satisfies (i) $(P' + k^+u) \cap S = \emptyset$ and (ii) $(P' + k^-u) \cap S = \emptyset$.

As the previous shots are hits, we also have $P' + (k^+ - 1)u \subset S$ and $P' + (k^- + 1)u \subset S$. According to the digital convexity of S, it follows from the two last inclusion that the convex hulls of these two sets are still included in S: $\mathrm{conv}(P' + (k^+ - 1)u) \cap \mathbb{Z}^2 \subset S$ and $\mathrm{conv}(P' + (k^- + 1)u) \cap \mathbb{Z}^2 \subset S$. With (i) and (ii), we obtain $(P' + k^+u) \cap \mathrm{conv}(P' + (k^+ - 1)u) = \emptyset$ and $(P' + k^-u) \cap \mathrm{conv}(P' + (k^- + 1)u) = \emptyset$. With translations, it leads to (iii) $(P' + u) \cap \mathrm{conv}(P') = \emptyset$ and (iv) $(P' - u) \cap \mathrm{conv}(P') = \emptyset$.

To arrive at a contradiction, we assume that there exists a pair of distinct points in the convex hull of P' and on the same Diophantine line parallel to u. In [4, 5], they show that a segment in direction u, included in the convex hull of P' and of maximal length has a vertex of the convex hull $a \in P'$ as an endpoint. Hence, we have that a is a vertex of the convex hull of P' and as the maximal length is at least $\|u\|$, either $a + u$ or $a - u$ is in the convex hull of P'. It contradicts either (iii) or (iv). ◁

Complexity Analysis

We prove Theorem 10 by computing the worst case number of misses of the width shooting algorithm. At the beginning, the set of the possible positions P is initialized as S. Its convex hull contains no more than the $n = |S|$ points of P. After the first step, according to Claim 11, the convex hull of the new set of positions P' has no more than one point per Diophantine line in the chosen direction u. It follows that the number of points of the convex hull of P' is less or equal to the number of Diophantine lines covering P which is $w(P) + 1$ since we choose the direction u providing this value. We use now the bound of Lemma 9: the inequality $n \geq w(S)^2/4$ leads to $w(S) + 1 \leq 2\sqrt{n} + 1$. For n larger than 25, we have $2\sqrt{n} + 1 \leq n^{3/4}$. It means that except if the number of possible positions falls under 25, the convex hull of the new set of possible positions contains fewer than $n^{3/4}$ lattice points.

By iterating k times the procedure (each time uses 2 misses), we have a set of possible positions whose convex hull contains at most $n^{(3/4)^k}$ points. The number of iterations k to get to fewer than 25 points is hence $O(\log\log n)$. Since each iteration has a constant number of misses, the total number of misses is also $O(\log\log n)$.

5 Conclusion and Open Problems

A simplified version of the children's game Battleship leads to numerous nontrivial questions and algorithms that we had a lot of fun to work on. We worked on the digital version of the problem, which is directly connected to the actual game. However, a continuous variation may also raise interesting questions.

Let $S \subset \mathbb{R}^2$ be a convex body. In the continuous version, instead of querying a point, the player can shoot along a ray until finding a miss (ray-shooting queries). The information that the player gets is the position of the boundary point of S on the ray. The problem consists of recovering the unknown position p of S with a small number of shots and it can be solved in the following manner. Let v denote the direction of a diameter of S. We shoot in directions v and $-v$ from the origin and determine with two queries the length of the line segment in direction v that passes through the origin. Since the direction v is a diameter direction, it is not possible to hit two parallel edges in the ray-shooting queries. Hence, there are at most two points that can give the same two results from the ray shooting queries. A third and last query is sufficient to distinguish these two points.

We conclude by listing several questions that remain open.

1. Given a finite shape $S \subset \mathbb{Z}^2$, what is the complexity to actually calculate its Battleship complexity $\mathrm{c}(S)$? Is the problem NP-complete as in the case of general minimal decision trees?
2. We consider the greedy shooting heuristic choosing at each node the shot providing the minimal number of misses. Does the heuristic provide an $O(\log n)$ approximation to the minimum number of misses? Does there exist better approximation algorithms?
3. For the class of polyominoes that are not HV-convex, we can build examples showing that the Battleship complexity is $\Omega(\log n)$. However, there is still a big gap with the upper bound of $n-1$. Can this gap be reduced?
4. For the class of digital convex sets, we provide an algorithm with at most $O(\log \log n)$ misses but in practice, the largest Battleship complexity that we found with the heuristic is only 3 as for continuous shapes. Is it possible that the Battleship complexity of the digital convex sets is also bounded by a constant?
5. Are there some other interesting classes of lattices sets for which efficient shooting algorithms can be found?
6. Lemma 8 states the inequality $n \geq \frac{3}{8}w(S)^2 - \frac{1}{2}w(S) + 3$ providing a lower bound on the number of points n of a digital convex set S according to its lattice width $w(S)$. This discrete version of Blaschke-Lebesgue inequality is however not tight. What is the best bound that can be achieved?
7. We defined the Battleship complexity in terms of the maximum number of misses. We could define an average version of the complexity, in which the average of $\mathrm{m}(x)$ for $x \in S$ is considered instead. What can be said about the average complexity?
8. If we count the total number of shots, instead of the number of misses we can adapt the algorithm from Section 4.2 to obtain an algorithm for digital convex shapes that uses $O(\log n)$ shots and that is optimal. What is the complexity of this variation for HV-convex polyominoes?
9. What is the complexity of the continuous version for different classes of shapes if both rotations and translations are allowed? Is it still constant for convex shapes?

References

1 Micah Adler and Brent Heeringa. Approximating optimal binary decision trees. In *11th Approximation, Randomization and Combinatorial Optimization, APPROX 2008*, pages 1–9, 2008. doi:10.1007/978-3-540-85363-3_1.

2 Esther M. Arkin, Michael T. Goodrich, Joseph S. B. Mitchell, David M. Mount, Christine D. Piatko, and Steven Skiena. Point probe decision trees for geometric concept classes. In *Third Workshop on Algorithms and Data Structures, WADS 1993*, pages 95–106, 1993. doi:10.1007/3-540-57155-8_239.

3 Esther M. Arkin, Henk Meijer, Joseph S. B. Mitchell, David Rappaport, and Steven Skiena. Decision trees for geometric models. In *Nineth Annual Symposium on Computational Geometry, SoCG 1993*, pages 369–378, 1993. doi:10.1145/160985.161167.

4 Imre Bárány and Zoltan Füredi. On the lattice diameter of a convex polygon. *Discrete Mathematics*, 241(1):41–50, 2001. Selected Papers in honor of Helge Tverberg. doi:10.1016/S0012-365X(01)00145-5.

5 Imre Bárány and Janos Pach. On the number of convex lattice polygons. *Combinatorics, Probability and Computing*, 1(4):295–302, 1992. doi:10.1017/S0963548300000341.

6 Loïc Crombez, Guilherme D. da Fonseca, and Yan Gérard. Efficient algorithms to test digital convexity. In *International Conference on Discrete Geometry for Computer Imagery, DGCI 2019*, pages 409–419, 2019. doi:10.1007/978-3-030-14085-4_32.

7 Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976. doi:10.1016/0020-0190(76)90095-8.

8 Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-sum and related problems. *Journal of the ACM*, 66(3), 2019. doi:10.1145/3285953.

9 Lior Rokach and Oded Maimon. *Data Mining With Decision Trees: Theory and Applications.* World Scientific Publishing Co., 2nd edition, 2014.

A Phase Transition in Minesweeper

Ross Dempsey

Joseph Henry Laboratories, Princeton University, NJ, USA
sdempsey@princeton.edu

Charles Guinn

Joseph Henry Laboratories, Princeton University, NJ, USA
cguinn@princeton.edu

Abstract

We study the average-case complexity of the classic *Minesweeper* game in which players deduce the locations of mines on a two-dimensional lattice. Playing *Minesweeper* is known to be co-NP-complete. We show empirically that *Minesweeper* exhibits a phase transition analogous to the well-studied `SAT` phase transition. Above the critical mine density it becomes almost impossible to play *Minesweeper* by logical inference. We use a reduction to Boolean unsatisfiability to characterize the hardness of *Minesweeper* instances, and show that the hardness peaks at the phase transition. Furthermore, we demonstrate algorithmic barriers at the phase transition for polynomial-time approaches to *Minesweeper* inference. Finally, we comment on expectations for the asymptotic behavior of the phase transition.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Complexity of Games, Minesweeper

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.12

Acknowledgements The authors would like to thank Dumitru Caligaru and Katherine Xiang for interesting conversations and correspondence related to this project. We are also grateful to Anton Belov and Joao Marques-Silva for their open source GMUS extractor `MUSer2`.

1 Introduction

Minesweeper is a single-player game originally released by Microsoft as part of the Windows 3.1 operating system. Players are presented with a rectangular lattice of covered squares, and behind some of these squares lie mines. The object of the game is to uncover all empty squares while never uncovering a mine. Players deduce the locations of mines from numbers on the empty squares which indicate how many mines lie adjacent.

Minesweeper is famous as a mode of mindless procrastination. On the television show *The Office*, Jim Halpert quips “those mines aren’t going to sweep themselves” in reference to his boredom. This popular sentiment would seem to be at odds with complexity-theoretic studies of *Minesweeper*. In Section 2 we review earlier results which show that playing *Minesweeper* by logical inference is co-NP-complete [12], and deciding whether a *Minesweeper* board is consistent with some assignment of mine squares is NP-complete [10], or even Turing-complete for an infinite board size [9].

This discrepancy is remedied by classic results on phase transitions in NP-complete problems. While NP-complete problems are widely believed to be exponentially hard in the worst case, they can be much easier in typical cases. As a representative example, for random k-`SAT` with n literals and c clauses, it is found empirically that the probability $p(n, c)$ of a formula being satisfiable drops sharply to zero when the ratio $\ell = c/n$ reaches a certain threshold [8]. In fact, Friedgut’s theorem [7, 6] implies the existence of a function $\ell_*(n)$ such that for any $\epsilon > 0$, as $n \to \infty$

$$p\left(n, (1-\epsilon)\ell_*(n)n\right) \to 1, \qquad p\left(n, (1+\epsilon)\ell_*(n)\right) \to 0. \tag{1}$$

It is suspected that $\ell_*(n)$ converges to a limiting critical value.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 12; pp. 12:1–12:10
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This phase transition is closely associated with the average-case complexity of `SAT`, or NP-complete problems more generally. Backtracking-based search algorithms for solving `SAT` require the largest number of branches for ℓ near ℓ_* [8], and a similar increase in hardness at the phase transition is observed in other constraint satisfaction problems [1]. As long as ℓ is sufficiently separated from ℓ_*, `SAT` can be solved relatively quickly in average cases.

In this paper, we present evidence for a similar phase transition in *Minesweeper* by showing empirically that the number of mines which can be flagged by an inference-based solver rapidly declines near a critical mine density. Furthermore, in Section 3 we define a metric for the hardness of *Minesweeper* instances and show that this metric peaks at the transition point. In Section 4, we discuss a simple family of polynomial-time algorithms and show that the hardness metric we employ represents a natural barrier for these algorithms. Finally, in Section 5, we comment on expectations for the asymptotic behavior of the phase transition.

2 Worst-Case Complexity: `CONSISTENCY` and `INFERENCE`

We begin with some definitions. Throughout this paper, when we refer to *Minesweeper* we mean the classic Windows game instantiated on an $N \times N$ square lattice. The rules of the game are as follows.

▶ **Definition 1.** *A* Minesweeper *board with size N and mine density ρ is an $N \times N$ array with $\lfloor N^2 \rho \rfloor$ sites occupied by mines, and the remainder of sites empty or "safe." A* Minesweeper *instance is a* Minesweeper *board with a subset of its sites uncovered. All the uncovered sites are labeled by the number of neighboring sites (including diagonal neighbors, and including wrap-around neighbors for sites on the edge) which contain mines. A move consists of one of the following:*

- *Revealing a covered site. This results in loss of the game if the site contains a mine. Otherwise, the site is uncovered and its number of neighboring mines is revealed. If this number is zero, all neighboring sites are revealed for the player automatically.*
- *Flagging a covered site. This is a bookkeeping tool for sites which are known to contain mines.*

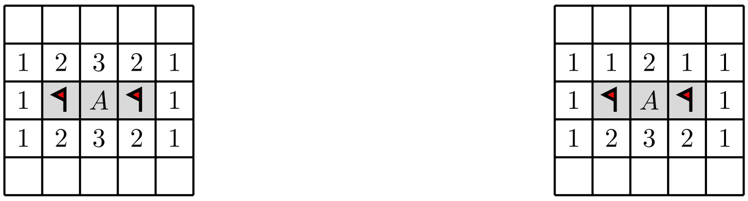
The game is won when all empty sites are uncovered.

When attempting to make inferences about the covered squares, it is helpful to adjust the labels of uncovered squares by subtracting the mines already accounted for with flags. We thus define the following [2]:

▶ **Definition 2.** *The effective label of a* Minesweeper *site its is label minus its number of flagged neighbors.*

Two decision problems related to *Minesweeper* have been investigated in the past. The first, historically speaking, is `CONSISTENCY` [10]. The `CONSISTENCY` problem asks whether a given *Minesweeper* instance could be realized by some assignment of mines to the covered squares. Figure 1b shows an example of an inconsistent instance which could never arise in an actual game of *Minesweeper*.

Clearly `CONSISTENCY` belongs to NP by reduction to `SAT`. We represent each covered site by a Boolean variable which is true if and only if the site contains a mine. Each site with at least one covered neighbor defines a Boolean constraint. The instance is consistent if and only if the conjunction of all these constraints forms a satisfiable formula [10].



(a) A consistent *Minesweeper* instance. The square marked A must contain a mine.

(b) An inconsistent *Minesweeper* instance. The uncovered sites above the flags imply $\neg A$, while those below the flags imply A.

Figure 1

In fact, `CONSISTENCY` is NP-complete. This is shown by reduction from `SAT`, via the implementation of logic gates as *Minesweeper* configurations which logically link the Boolean variables stored at the covered sites. For example, an AND gate would be constructed by setting up a *Minesweeper* instance with two covered sites a and b, and a third covered site c, with labels on the uncovered sites which together imply that c contains a mine only if both a and b contain mines. Detailed constructions of such an AND gate as well as a NOT gate are known [10], and with these components any Boolean circuit can be built. One can construct arbitrarily complex circuits on an infinite *Minesweeper* board, and so `CONSISTENCY` on an infinite board is Turing-complete [9].

As the authors of [12] note, solving `CONSISTENCY` is not directly relevant to a *Minesweeper* player, who is promised a consistent configuration. Instead, the *Minesweeper* player is tasked with deciding whether there exists a covered square which can be inferred to contain a mine, or to not contain a mine. We call this problem `INFERENCE`.

There is a simple reduction from `INFERENCE` to the complement of `CONSISTENCY`. Given a *Minesweeper* instance, we iterate over all the covered sites and tentatively assume they are either empty or contain mines. For each of these tentative assignments, we consult a `CONSISTENCY` oracle; if the oracle ever tells us we have an inconsistent board, then our tentative assignment is incorrect and we make the opposite inference. Since `CONSISTENCY` is in NP, this reduction shows that `INFERENCE` is in co-NP. Moreover, by reduction from `UNSAT`, it has been shown that `INFERENCE` is co-NP-complete [12].

3 Phase Transition

If P $\neq$ NP, the results of Sec. 2 imply that no polynomial-time algorithm can decide `INFERENCE` for all instances. Nonetheless, specific instances of *Minesweeper* may yield to a polynomial-time approach, and experience suggests that this is the case for sufficiently low mine density. By contrast, at sufficiently high mine density, we expect to encounter configurations like that in Figure 2 where no inference is possible.

For intermediate densities, then, *Minesweeper* must go from being easily solvable by inference to rarely solvable by inference. This claim is simple to test empirically. We start with *Minesweeper* boards with a single site guaranteed to the player to be labeled zero, in order to remove uncertainty arising from the first move [2]. We use the reduction from `INFERENCE` to `UNSAT` outlined in the previous section to play *Minesweeper*, deducing an empty or mined site at each step until either the game is won or no more inferences are possible. Our algorithm is outlined as follows.

1. Start by uncovering the guaranteed 0-labeled site.
2. Collect all covered squares bordering uncovered squares into a list `frontier_outer`, and all uncovered squares bordering covered squares into a list `frontier_inner`.

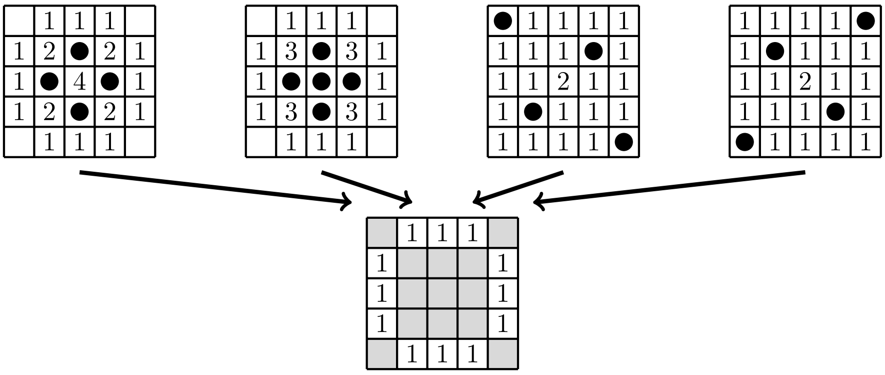


Figure 2 No inference is possible in the *Minesweeper* instance in the bottom panel, since the instance could have arisen from any of the boards shown above (among others), and for each covered site there is a consistent configuration in which it contains a mine and another in which it is empty. Note that we are not employing periodic boundary conditions here; alternatively, this grid should be thought of as part of a large *Minesweeper* instance.

3. Assign a Boolean variable to each site in `frontier_outer`.
4. From each site i in `frontier_inner`, construct a Boolean constraint F_i, and transform it into conjunctive normal form (CNF). For example, if a site has effective label 1 and borders two covered sites assigned to variables x_β and x_γ in the previous step, then we have

   $$(\neg x_\beta \wedge x_\gamma) \vee (x_\beta \wedge \neg x_\gamma), \tag{2}$$

   which can be written in CNF as

   $$F_i = (x_\beta \vee x_\gamma) \wedge (\neg x_\beta \vee \neg x_\gamma). \tag{3}$$

5. Let $F = \bigwedge_{i \in I} F_i$, with I an index set for `frontier_inner`, be the conjunction of all the formulas constructed in the previous step. For every site in `frontier_outer`, with Boolean variable x_β, test $F \wedge x_\beta$ for satisfiability. If it is unsatisfiable, then this site must not have a mine, so reveal it. If it is satisfiable, then test $F \wedge \neg x_\beta$ for satisfiability. If it is unsatisfiable, then this site must have a mine, so flag it.
6. If all mines are flagged, or if no inferences are found, conclude. Otherwise, return to step 2.

In Figure 3, we show the results of these tests. For grids of sizes $N = 20$, 40, and 80, we plot the empirical fraction α of mines flagged by inference. We find that α rapidly declines to 0 roughly in the interval $0.2 < \rho < 0.3$, and the decline is steeper at higher values of N.

We would also like to determine how difficult it is to solve the *Minesweeper* inference problem at various mine densities. We use an algorithm-agnostic metric derived from the reduction to `UNSAT`. As described in the steps above, we construct a Boolean formula F_i from every labeled site in the inner frontier. We then test the conjunction of all these formulas, together with one tentative assumption:

$$\bigwedge_{i \in I} F_i \wedge x_\beta, \qquad \text{or} \qquad \bigwedge_{i \in I} F_i \wedge \neg x_\beta, \tag{4}$$

where β is the index of a covered site on the outer frontier and I indexes the inner frontier. If one of these formulas is unsatisfiable, then we have proved by contradiction the presence

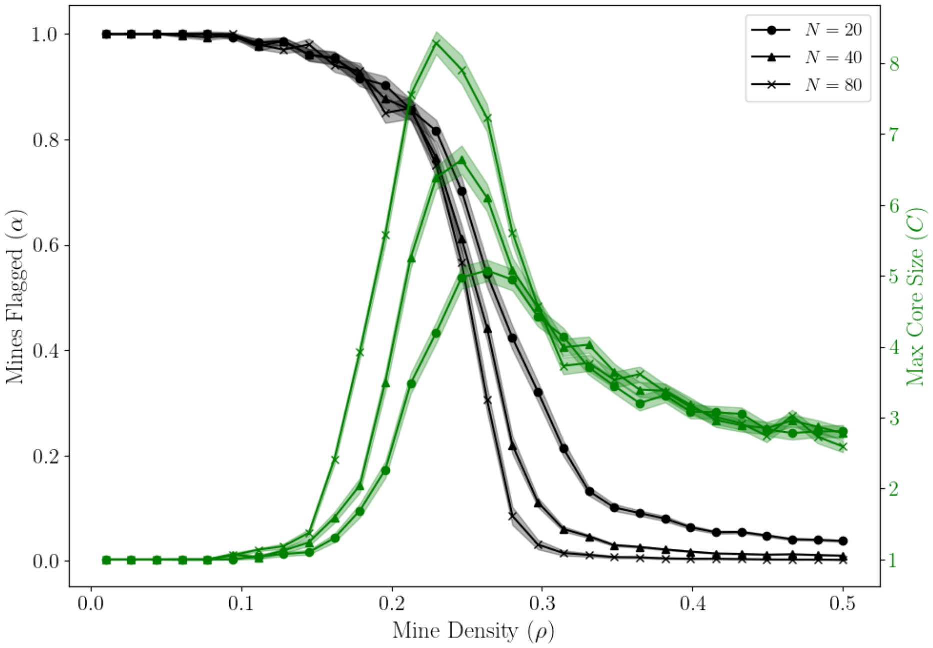


Figure 3 Averaged results and standard errors of 300 *Minesweeper* games at each mine density for each grid size $N = 20$, 40 and 80 are shown. In black, we plot the expected fraction α of mines flagged by the INFERENCE solver as a function of the mine density ρ. In green, we plot the expected maximum size of the grouped minimal unsatisfiable (GMUS) cores encountered in the reduction to SAT. This is a measure of the hardness of *Minesweeper* as a function of ρ, and roughly indicates the number of labeled sites one needs to look at in order to make inferences.

or absence of a mine. In this case we can extract the *grouped minimal unsatisfiable (GMUS) core*, a subset $S \in I$ for which $\bigwedge_{i \in S} F_i \wedge x_\beta$ is unsatisfiable, but for every proper subset $S' \subsetneq S$ the formula $\bigwedge_{i \in S'} F_i \wedge x_\beta$ is satisfiable (or the equivalent with $\neg x_\beta$ in place of x_β). As a measure of hardness we use $C = |S|$, the number of labeled sites one needs to look at in order to make the inference about the presence or absence of a mine at site β.

Figure 3 shows the average value of the maximum GMUS core size encountered during the *Minesweeper* games as a function of the mine density. This is computed using the open-source MUSer2 library [3]. The core size is strongly peaked in the vicinity of the phase transition, and the peak core size grows with the size of the lattice. This indicates that playing *Minesweeper* near the critical mine density requires looking at large patches of the board to make inferences. Furthermore, we clearly see the easy-hard-easy pattern first suggested in [4].

4 Algorithmic Barriers

In Section 3 we showed that the maximum GMUS core size encountered during the course of a *Minesweeper* game peaks at the phase transition. This suggests that the hardest *Minesweeper* instances arise at the critical mine density. As another test of this hypothesis, we present a family of polynomial-time approaches to INFERENCE. These algorithms partially solve INFERENCE, in the sense that they can identify some but not all inferences, and never make an invalid inference; there are false negatives, but no false positives. We show that they perform well below the critical mine density, but fail to find a significant proportion of inferences at or near the critical density.

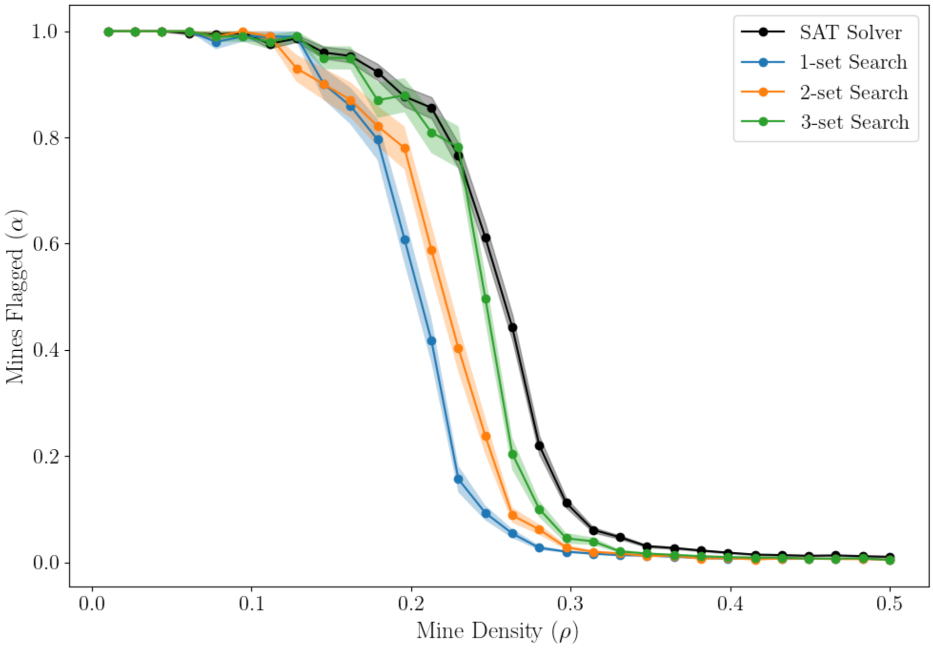


Figure 4 The performance of the SAT solver is compared to the polynomial-time k-set search algorithm on a board of size $N = 40$. The SAT solver finds the theoretical maximum fraction of mines which can be flagged by inference. The k-set search approach performs comparably until the phase transition at $0.2 < \rho < 0.3$, at which point the SAT solver clearly outperforms k-set search, and moreover the polynomial-time algorithms are stratified by k.

Our algorithms are a generalization of the "naïve single-point" algorithm in [2]. In the naïve single-point approach, only two types of inferences can be made:

- If the effective label of a site is 0, then any of its covered neighbors can be inferred to be safe.
- If the effective label of a site is equal to its number of covered neighbors, then any of its covered neighbors can be inferred to be mines.

These sorts of inferences are possible precisely when $C = 1$, i.e., when the SAT solver need only use a single labeled site to make an inference.

The naïve single-point algorithm has $\mathcal{O}\left(n_{\text{inner}}\right)$ complexity on a single *Minesweeper* instance, where n_{inner} is the size of the inner frontier. We employ a natural generalization with $\mathcal{O}\left(n_{\text{inner}}^k\right)$ complexity, which we call k-set search. We label the sites of the outer frontier by $j = 1, \ldots, n_{\text{outer}}$, and assign to each site $i = 1, \ldots, n_{\text{inner}}$ in the inner frontier a vector

$$a_{ij} = \begin{cases} 1 & \text{if } i \text{ borders } j \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

and denote the effective label of site i with e_i. In this notation, the constraint provided by a site in the inner frontier is

$$\sum_{j=1}^{n_{\text{outer}}} a_{ij} x_j = e_i. \tag{6}$$

We then iterate over all combinations of up to k of these vectors, with replacement, denoted by index sets $\{i_1, \ldots, i_k\}$. For each such combination, we consider all the linear combinations of constraints,

$$\sum_{\ell=1}^{k}(-1)^{b_\ell}\sum_{j=1}^{n_{\text{outer}}}a_{i_\ell j}x_j=\sum_{\ell=1}^{k}(-1)^{b_\ell}\,e_{i_\ell}, \tag{7}$$

with $b_\ell = 0, 1$. If the right hand side is equal to either the minimum or maximum value of the left hand side, then we can infer the value of any of the variables appearing on the left hand side with a nonzero coefficient.

It is simple to see that any inference made by k-set search could also be made by a `SAT` solver limited to a GMUS core size $C \leq k$. The converse is almost true: if the GMUS core has size k, k-set search will only fail if the inference comes from a linear combination of constraints with coefficients other than ±1, which is rare. We thus expect that k-set search will perform well up until the point where the core size reaches k. Figure 3 shows that the core size is peaked around the phase transition, so in practice we expect that k-set search will perform significantly less well than a full `SAT` solver in the vicinity of the phase transition.

Figure 4 confirms this claim. We plot the average fraction α of mines flagged by 1-, 2-, and 3-set search, compared to the performance of the `SAT` solver. As we expect, all algorithms perform roughly equally at low ρ, and then stratify by k in the vicinity of the phase transition. In the interval $0.2 < \rho < 0.3$, a *Minesweeper* player would benefit substantially from using $(k + 1)$-set search over k-set search.

5 Asymptotic Behavior

Naturally, one of the key questions regarding the *Minesweeper* phase transition is its asymptotic behavior. Our empirical studies strongly suggest some kind of phase transition, loosely speaking, as $N \to \infty$. The question is whether this transition is sharp. Let $\alpha(N, \rho)$ denote the expected fraction of mines which can be flagged by inference on a board of size N with mine density ρ, as in Figure 3. Following [7], we say the *Minesweeper* transition is coarse if there exists some constant $C > 0$ such that, for all ρ,

$$\lim_{N\to\infty}\left|\rho\frac{d\alpha(N,\rho)}{d\rho}\right| \leq C. \tag{8}$$

Otherwise, we say the transition is sharp.

For several reasons, it is difficult to directly address the question of whether *Minesweeper* has a sharp phase transition. First, the distribution of *Minesweeper* instances is quite complicated. Although it is simple to define a uniformly random distribution of *Minesweeper* boards, the *Minesweeper* instances which arise during game-play depend on a player's strategy.

Even if we restrict to the case of the perfectly logical player who operates by solving `UNSAT`, the frontiers which arise depend on the geometry of the percolation clusters formed by the mines and the sites with nonzero labels. The probability of a given site having either a mine or a nonzero label is

$$P\,(\text{mine or nonzero}) = 1 - P(\text{zero}) = 1 - (1-\rho)^9. \tag{9}$$

Setting this equal to the threshold value of $p \approx 0.59$ for site percolation on a 2D square lattice [13], we find $\rho \approx 0.1$. However, we are not justified in making such a direct comparison to the standard percolation problem, because nearby sites in *Minesweeper* are correlated. To address this, we perform Monte Carlo simulations of the formation of percolation clusters in

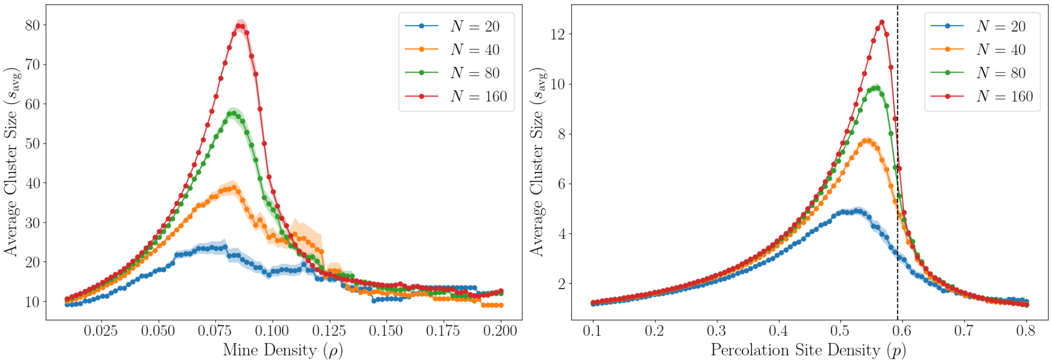


Figure 5 The average cluster sizes appearing in the standard percolation problem (right) for a range of values of the site probability, and in the sites with mines or nonzero labels in *Minesweeper* (left) for a range of values of the mine density. The vertical line on the right shows the threshold probabiilty for percolation.

Minesweeper. Figure 5 shows the average cluster size s_{avg}, defined as in [5], of the clusters which form on *Minesweeper* boards compared to those in standard percolation theory. The behavior at $\rho \approx 0.1$ shows that our calculation gives roughly the correct location of the percolation threshold for *Minesweeper*.

The geometry of the percolation clusters in the vicinity of the *Minesweeper* percolation threshold is related to that of the clusters in standard percolation theory with independent sites. This follows from the general principle of universality: the “microscopic” details of a system do not affect bulk properties at the critical point. More precisely, renormalization group methods show that if correlations between sites fall off faster than r^{-d}, then the correlations are irrelevant at the critical point as long as $d\nu > 2$, where ν is the critical exponent for correlation length [11]. The correlations in *Minesweeper* have finite range, so they decay faster than any power law, and hence we can ignore the correlations when considering the geometry of the frontier near the percolation threshold. It is well-known that percolation clusters exhibit complex fractal geometry near the percolation threshold [14], and so this complexity should carry over to random *Minesweeper* instances.

All this is simply to argue that random *Minesweeper* instances sample in a very complicated way from the space of possible `INFERENCE` problems. There are thus two questions of separate import. First we should ask if `INFERENCE` itself, or some generalization of it, has a sharp phase transition with respect to some parametrized distribution of instances. If this is answered in the affirmative, then the next question is whether *Minesweeper* itself samples from the space of `INFERENCE` problems in such a way as to lead to a phase transition with respect to the mine density ρ.

It seems likely that `INFERENCE` itself exhibits a phase transition. To see this, we consider a space of problems which contains `INFERENCE` as a special case, and for which it is much simpler to define random instances. We start by noting that (5) defines the incidence matrix of a hypergraph where vertices correspond to covered sites in the outer frontier and hyperedges correspond to labeled sites in the inner frontier. Each hyperedge can be assigned the effective label of its corresponding site, and then `INFERENCE` becomes the following question: if each vertex must be assigned a value 0 or 1, such that the sum of vertices in each hyperedge is equal to its label, is the value of any vertex constrained to be either 0 or 1?

The natural generalization is to replace the hypergraph defined by (5) with an arbitrary hypergraph, and the effective labels with an integer label between 0 and $|e|$ on each hyperedge e. We may define a probability measure μ on hypergraphs for which hyperedges of size m appear independently with probability p_m.

The presence of a sharp phase transition in properties of graphs or hypergraphs can be shown using Friedgut's theorem [6]. Roughly speaking, this theorem establishes that any property of hypergraphs which does not exhibit a sharp phase transition must be approximable by a local property. We will not prove the existence of a sharp phase transition for inference on random hypergraphs, but intuitively, it is clear that the inference property should not be locally approximable. Inferences can arise from large sets of hyperedges in myriad ways; we cannot expect to account for almost all inferences by checking for a finite set of sub-hypergraphs. Indeed, the k-set search algorithm of Section 4 effectively checks for a finite set of sub-hypergraphs, and for any fixed k we do not expect this algorithm to perform well for sufficiently large N. These considerations suggest that inference on random hypergraphs has a sharp phase transition at some hyperedge probability $p_*(N)$.

Returning to the question of the *Minesweeper* phase transition, some simple estimates suggest that even if inference on random hypergraphs does have a sharp phase transition, the behavior of the *Minesweeper* phase transition depends on details which are difficult to ascertain. The distribution of hypergraphs which appear in the inference problems in *Minesweeper* is certainly a complicated function of the mine density, and even for fixed mine density, overlapping hyperedges are not independent of one another. Thus, throughout a game of *Minesweeper*, the `INFERENCE` problems encountered by a player are sampled from distributions of hypergraph inference problems with a range of parameters.

As a simple phenomenological model of this behavior, assume that the `INFERENCE` problems which appear during a game of *Minesweeper* are sampled using some parameter p, and that there is a sharp threshold at $p = p_*(N)$ above which inferences become possible. Furthermore, assume the parameter p is itself drawn from some distribution $f(p; N, \rho)$. The game of *Minesweeper* will continue until the random value of p falls below $p_*(N)$. On each turn, this happens with probability

$$P(N,\rho) = \int_0^{p_*(N)} f(p; N, \rho)\, dp. \tag{10}$$

The expected fraction of mines flagged then scales as the number of turns played, so

$$\alpha(N,\rho) \sim \frac{1}{N} \sum_{n=0}^{T(N,\rho)} n(1-P(N,\rho))^n P(N,\rho) \sim (1-P(N,\rho))^{T(N,\rho)}, \tag{11}$$

where $T(N,\rho)$ is the expected number of turns required to complete the game if all mines are flagged. As ρ is increased, we expect P to increase, since under-constrained inference problems like the one in Figure 2 are more likely to appear. Thus, (11) reproduces the sigmoidal behavior evident in Figure 3.

As N is taken to infinity, the shape of $\alpha(N,\rho)$ depends on the exact limiting behavior of $P(N,\rho)$ and $T(N,\rho)$, which in turn depend on the detailed statistics of *Minesweeper* instances. It may approach either a discontinuous function, corresponding to a sharp phase transition, or some curve which interpolates between $\alpha = 1$ and $\alpha = 0$ over a finite range of mine densities, giving a coarse phase transition.

Despite this uncertainty in the asymptotic behavior, our empirical results show that the phase transition in *Minesweeper* at finite N is closely associated with the appearance of hard `INFERENCE` problems. We can thus understand a great deal about the average-case complexity of *Minesweeper* by thinking in terms of its phase transition.

References

1 Dimitris Achlioptas and Amin Coja-Oghlan. Algorithmic barriers from phase transitions. *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, October 2008. doi:10.1109/focs.2008.11.

2 David J Becerra. Algorithmic approaches to playing Minesweeper. Master's thesis, Harvard University, 2015.

3 Anton Belov and Joao Marques-Silva. MUSer2: an efficient MUS extractor, system description. *JSAT*, 2012.

4 Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

5 Kim Christensen. Percolation theory. *Imperial College London*, 1, 2002.

6 Ehud Friedgut. Hunting for sharp thresholds. *Random Structures & Algorithms*, 26(1-2):37–51, 2005.

7 Ehud Friedgut, Jean Bourgain, et al. Sharp thresholds of graph properties, and the k-SAT problem. *Journal of the American Mathematical Society*, 12(4):1017–1054, 1999.

8 Ian P. Gent and Toby Walsh. The SAT phase transition. In *ECAI*, 1994.

9 R. Kaye. Infinite versions of Minesweeper are Turing complete. URL: http://web.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf.

10 R. Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22:9–15, 2000.

11 Abbas Ali Saberi. Recent advances in percolation theory and its applications. *Physics Reports*, 578:1–32, May 2015. doi:10.1016/j.physrep.2015.03.003.

12 Allan Scott, Ulrike Stege, and Iris van Rooij. Minesweeper may not be NP-complete but is hard nonetheless. *The Mathematical Intelligencer*, 33:5–17, 2011.

13 Dietrich Stauffer and Ammon Aharony. *Introduction to percolation theory.* Taylor & Francis, 1994.

14 Yakov M. Strelniker, Shlomo Havlin, and Armin Bunde. Fractals and percolation. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 3847–3858. Springer New York, New York, NY, 2009. doi:10.1007/978-0-387-30440-3_227.

On the Treewidth of Hanoi Graphs

David Eppstein
University of California, Irvine, CA, USA
eppstein@uci.edu

Daniel Frishberg
University of California, Irvine, CA, USA
dfrishbe@uci.edu

William Maxwell
Oregon State University, Corvallis, OR, USA
maxwellw@oregonstate.edu

Abstract

The objective of the well-known *Towers of Hanoi* puzzle is to move a set of disks one at a time from one of a set of pegs to another, while keeping the disks sorted on each peg. We propose an adversarial variation in which the first player forbids a set of states in the puzzle, and the second player must then convert one randomly-selected state to another without passing through forbidden states. Analyzing this version raises the question of the *treewidth* of *Hanoi graphs*. We find this number exactly for three-peg puzzles and provide nearly-tight asymptotic bounds for larger numbers of pegs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Hanoi graph, Treewidth, Graph separators, Kneser graph, Vertex expansion, Haven, Tensor product

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.13

Related Version A full version of the paper is available at `https://arxiv.org/abs/2005.00179`.

Funding This research was supported in part by NSF grants CCF-1618301, CCF-1616248, and CCF-1617951.

Acknowledgements Some of the results in the section on three-peg Hanoi graphs were previously announced on a web forum [7].

1 Introduction

The *Towers of Hanoi* puzzle is very well known (for a comprehensive treatment see [10]), but it loses its fun once its player learns the strategy. It has some number n of disks of distinct sizes, each with a central hole allowing it to be stacked on any of three pegs. The disks start all stacked on a single peg, sorted from largest at the bottom to smallest at the top. They must be moved one at a time until they are all on another peg, while at all times keeping the disks in sorted order on each peg. The optimal strategy is easy to follow: alternate between moving the smallest disk to a peg that was not its previous location, and moving another disk (the only one that can be moved). Once one learns how to do this, and that the strategy takes $2^n - 1$ moves to execute [18], it becomes tedious rather than fun.

The puzzle can be modified in several ways to make it more of an intellectual challenge and less of an exercise in not losing one's place. One of the most commonly studied variations involves using some number p of pegs that may be larger than three. Of course, one can ignore the extra pegs, but using them allows shorter solutions. An optimal solution for four pegs was given by Bousch in 2014 [5], but the best solution for larger numbers of pegs remains open. The Frame–Stewart algorithm solves these cases, but it is not known if it is

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 13; pp. 13:1–13:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

optimal [20]. The length of an optimal solution, for starting and ending positions of the disks chosen to make this solution as long as possible, can be modeled graph-theoretically using a graph called the *Hanoi graph*, which we denote H^n_p. This graph is formed by constructing a vertex for each configuration of the game, and connecting two vertices with an edge when their configurations are connected by one legal move. The number of moves between the two farthest-apart positions is then the diameter of this graph. For three pegs, the diameter of H^n_3 is $2^n - 1$ (the traditional starting and ending positions are the farthest apart) but for $p > 3$ the diameter of H^n_p is unknown [13].

In this paper, we consider a different way of making the puzzle more difficult, by making it adversarial. In our version of the game, the first of two players selects a predetermined number of *forbidden positions*, that the second player cannot pass through. Then, the second player must solve a puzzle using the remaining positions. If that were all, then the first player could win by forbidding only a very small number of positions, the $p - 1$ positions one move away from the start position. To make the first player work harder, after the first player chooses the forbidden positions, we choose the start and end position randomly from among the positions in the game. We ask: How many positions must the first player forbid, in order to make this a fair game, one where both players have equal chances of being able to win?

We can model this problem graph-theoretically, as asking for the smallest number of vertices to remove from a Hanoi graph in order for the number of pairs of remaining vertices belonging within the same component as each other to be half the total number of pairs of vertices. The answer to the problem lies between the minimum size of a *balanced vertex separator* (Lemma 2) and (up to a constant factor of three) the minimum order of a *recursive balanced vertex separator*; the latter is equivalent, up to constant factors, to asking for the *treewidth* of H^n_p. (Technically, the treewidth can be larger than the recursive separator order by a logarithmic factor when this order is constant, but both are within constant factors of each other when the order is polynomial.) Treewidth is of interest to computer scientists as many NP-hard graph problems become fixed-parameter tractable on graphs with bounded treewidth [4].

1.1 New results and prior work

We conjecture that the treewidth of H^n_p is $\Theta((p-2)^n)$. For $p > 3$ this bound is exponential, and we make progress towards this conjecture by proving that the treewidth is within a polynomial factor of this bound. More precisely we show an asymptotic upper bound of $O((p-2)^n)$ and an asymptotic lower bound of $\Omega(n^{-(p-1)/2} \cdot (p-2)^n)$. We increase the lower bound to $\Omega(\frac{2^n}{n})$ when $p = 4$. Moreover, we find the exact (constant) treewidth of H^n_3 and of the closely-related Sierpínski graphs. Our results provide an answer to our motivating question on sizes of forbidden sets of positions, up to polynomial factors for four or more pegs and exactly for three pegs.

As a byproduct of our proof techniques, we observe a nearly linear asymptotic lower bound on the treewidth of the *Kneser graph* (Corollary 25). Harvey and Wood [12] showed a previous exact result for the treewidth of $\mathrm{Kn}(n, k)$ when n is at least quadratic in k. Another byproduct of our proof techniques gives a new lower bound on the treewidth of the *tensor product* $G \times H$ of two graphs G and H, when H is not bipartite. Eppstein and Havvaei [8] gave an upper bound on the treewidth of $G \times H$; Brevšar and Spacapan [6] gave an analogous lower bound for edge connectivity; Kozawa et al. [14] gave lower bounds for the treewidth of the strong product and Cartesian product of graphs.

2 Preliminaries

2.1 Hanoi graphs

Label the n disks of the Towers of Hanoi, in order of increasing size, as $d_1, \ldots, d_n$. If disks d_i and d_j are on the same peg, and $i < j$, then d_j is constrained to be below d_i. A legal move in the game consists of moving the top (smallest) disk on some peg A to another peg B, while preserving the constraint. At the beginning of the game, all n disks are on the first peg. The objective of the game is to obtain, through some sequence of legal moves, a state in which all n disks are on the last peg. Let p be the number of pegs. Traditionally, $p = 3$.

Formally, a *configuration* of the p-peg, n-disk Towers of Hanoi game is an n-tuple $(p_1, p_2, \ldots, p_n)$ where $p_i \in \{1, 2, \ldots, p\}$, describing the peg for each disk d_i. We say two configurations $(p_1, p_2, \ldots, p_n)$ and $(p'_1, p'_2, \ldots, p'_n)$ are *compatible* if a move from one configuration to the other is allowed. This happens exactly when the two configurations differ only in the value of a single coefficient p_i, for which d_i is the smallest disk having either of the two differing values. We call a configuration with each disk on the same peg a *perfect state*. The *Hanoi graph* H_p^n is a graph whose vertices are the configurations of the n-disk, p-peg Towers of Hanoi game, with an edge for each compatible pair of configurations. It has p^n vertices and $\frac{1}{2}\binom{p}{2}(p^n - (p-2)^n)$ edges [1].

2.2 Recursive balanced separators, treewidth, and havens

In this section we give a brief discussion of the concepts of recursive balanced separators, treewidth, and havens. Given a graph $G = (V, E)$ a *vertex separator* is a subset $X \subseteq V$ such that $G \setminus X$ consists of two disjoint sets of vertices A and B with $A \cup B = V \setminus X$ and for all $a \in A$, $b \in B$ there is no edge (a, b) in the graph $G \setminus X$. Further, given a constant c with $\frac{1}{2} \leq c < 1$, we call X a *balanced vertex separator* if $(1-c)|V| \leq |A| \leq \frac{|V|}{2}$ and $\frac{|V|}{2} \leq |B| \leq c|V|$. When this holds we call X a *c-separator*. We say that G has a *recursive balanced separator* of order s, where $s : \mathbb{N} \to \mathbb{N}$ is a nondecreasing function, whenever either $|V| \leq 1$, or we can find a balanced separator of size $s(|V|)$ for G, and the resulting subgraphs A and B have recursive balanced separators of order s respectively. We abuse notation and refer to $s(|V|)$ as $s(G)$.

A *tree decomposition* of a graph G is a tree T whose nodes are sets of vertices in G called *bags*, such that the following conditions hold.

- If two vertices are adjacent, then they share at least one bag.
- If a vertex v is in two bags A and B, then v is in every bag on the path from A to B in T.
- Every vertex in $V(G)$ is in some bag.

The *width* of a tree decomposition T is one less than the maximum size of a bag in T. The *treewidth* of a graph G, denoted $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of G. The bags in the tree decomposition T induce vertex separators in G. Moreover, we can use the tree decomposition to find a recursive balanced separator for G. Hence, the treewidth of G is a measure of the minimum order of a recursive balanced separator for G. The following folklore lemma relates the order of a recursive balanced separator to the treewidth of a graph; see [9] and [17, Lemma 6.6].

▶ **Lemma 1.** *Let G be an N-vertex graph. If $t = \mathrm{tw}(G)$, then with respect to every constant $\frac{1}{2} \leq c < 1$, G has a recursive balanced separator of order $s(N') = t + 1$ for all $1 \leq N' \leq N$. On the other hand, if G has a recursive balanced separator of order t, where $t = \Omega(N^d)$ for some constant $d > 0$, then G has treewidth $O(t)$.*

Returning to our motivating game, in which one player forbids the use of a designated set of states in the state space of a puzzle and the other player attempts to connect two randomly chosen states by a path, we see that a fair number of states to forbid is controlled by the size of a recursive balanced separator. We formalize this in the following lemma:

▶ **Lemma 2.** *Given a graph G, let $f(G)$ be the minimum number of vertices that can be removed from f so that, if two random vertices of G are chosen, the probability that they are not removed and have a path between them is at most $1/2$. Let $c = 1/\sqrt{2}$, and let $r(G)$ be the minimum size of a c-separator (not necessarily recursive) for G. Let s be the minimum order of a recursive c-separator for G. Then $r(G) \leq f(G) \leq 3s(G)$.*

Proof. If we remove a vertex set X with $|X| = f(G)$, leaving probability less than $1/2$ that two randomly-chosen vertices from G are connected, then the remaining subgraph cannot contain any connected component larger than $|V(G)|/\sqrt{2}$. If it contains any connected component of size at least $|V(G)|/2$, then f separates that subgraph from the remaining vertices, and otherwise the remaining small subgraphs can be combined to give a separation between two subgraphs whose largest size is at most $2|V(G)|/3$, better than c. Therefore, $r(G) \leq f(G)$.

To show that $f(G) \leq 3s(G)$, find a recursive c-separator for G of order s; the separator X has the following three separators as subsets: a c-separator X for G resulting in two separated subgraphs, and c-separators Y and Z for each of the two separated subgraphs. $|X|+|Y|+|Z| \leq 3s(G)$. Removing $X \cup Y \cup Z$ from G partitions the rest of G into subgraphs of size at most $|V(G)|/2$. No matter which of these subgraphs one of the randomly chosen two vertices belongs to, the probability that the other vertex belongs to the same component will be at most $1/2$. ◀

Some of our results will bound the treewidth of graphs using *havens*, a mathematical formalization of an escape strategy for a robber in cop-and-robber pursuit-evasion games. In these games, a set of cops and a single robber are moving around on a given graph G. Initially the robber is placed at any vertex of the graphs, and none of the cops has been placed. In any move of the game, one of the cops can be removed from the graph, or a cop that has already been removed can be placed on any vertex of the graph. However, before the cop is placed, the robber (knowing where the cop will be placed) is allowed to move along any path in the graph that is free of other cops. The goal of the cops is to place a cop on the same vertex as the robber while simultaneously blocking all escape routes from that vertex, and the goal of the robber is to evade the cops forever. In these games, a *haven of order k* describes a strategy by which the robber can perpetually evade k cops, by specifying where the robber should move for each possible move by the cops. It is defined as a function ϕ, mapping each set of vertices $X \subseteq V$ with $|X| \leq k$ to a nonempty connected component in $G \setminus X$, such that whenever $X_1 \subseteq X_2$, $\phi(X_2) \subseteq \phi(X_1)$. A robber following this strategy will move to any vertex of $\phi(X)$, where X denotes the set of vertices to be occupied by the cops at the end of the move. The mathematical properties of havens ensure that the robber can always reach one of these vertices by a cop-free path.

Returning again to our adversarial version of the Towers of Hanoi puzzle, the cops-and-robber game is equivalent to a game in which the first player attempts to pin the second player to a state from which no legal move to any non-forbidden state is possible. The placement (or removal) of a cop is equivalent to the first player designating (or de-designating) a state as forbidden; an evasion strategy for a robber is equivalent to the existence of a legal move for the second player.

The existence of a haven in G yields a lower bound on the treewidth of G via the following lemma.

▶ **Lemma 3** (Seymour and Thomas [19]). *A graph G has a haven of order greater than or equal to k if and only if* $\operatorname{tw}(G) \geq k - 1$.

3 Three pegs

In this section we show that $\operatorname{tw}(H_3^n) \leq 4$ for all $n \geq 1$. We prove this by relating the three-peg Towers of Hanoi game and the Sierpínski triangle graphs, which we denote S_n. S_n has treewidth at least 3 for all n, as it contains a triangle, and (Lemma 4) it equals 4 for $n > 4$. Additionally, each Sierpínski triangle graph contains a smaller three-peg Hanoi graph as a minor, and vice versa. From this it will follow that $\operatorname{tw}(H_3^n) = 4$ for all sufficiently large n. For completeness we include a more detailed proof of the bounds on $\operatorname{tw}(S_n)$.

We define the Sierpínski triangle graphs, along with a planar embedding of them, inductively. The planar embedding will allow us to see the geometric similarity between the Sierpínski graphs and the three-peg Hanoi graphs. The first Sierpínski triangle, S_1, is isomorphic to K_3 with a planar embedding of an equilateral triangle with unit length sides. The vertices of the triangle coincide with the vertices of K_3.

Inductively, we assume S_{n-1} has a planar embedding whose outer face is embedded geometrically as an equilateral triangle. We label the vertices on the outer face of the triangle v_ℓ, v_r, v_t which are the left, right, and top vertices, respectively. To construct S_n from S_{n-1} we take three copies of S_{n-1} labeled $S_{n-1}^L, S_{n-1}^R, S_{n-1}^T$ for the left, right, and top triangles and make the following vertex identifications.

1. Identify v_ℓ in S_{n-1}^R with v_r in S_{n-1}^L, and call the resulting vertex $v_{\ell r}$.
2. Identify v_t in S_{n-1}^L with v_ℓ in S_{n-1}^T, and call the resulting vertex $v_{\ell t}$.
3. Identify v_t in S_{n-1}^R with v_r in S_{n-1}^T, and call the resulting vertex v_{rt}.

The resulting graph has a planar embedding whose outer face can again be embedded as a subdivided equilateral triangle. In S_n the left, right, and top vertices of the outer face are contained in S_{n-1}^L, S_{n-1}^R, and S_{n-1}^T respectively. As before we denote them as v_ℓ, v_r, and v_t. Note that we can recursively decompose S_n into a triangle and a trapezoid, from which the trapezoid further decomposes into two additional triangles. (Here, we only consider trapezoids whose long side is horizontal.) This recursive decomposition leads to the construction of a tree decomposition of S_n. The six distinguished vertices $v_\ell, v_r, v_t, v_{\ell r}, v_{\ell t}$, and v_{rt} define the bags of the tree decomposition at each level. The set $\{v_t, v_{\ell t}, v_{rt}, v_\ell, v_r\}$ lies on the perimeter of a triangle in this decomposition. We call a bag in the tree decomposition consisting of these vertices a *triangular bag*. On the other hand, the set $\{v_{\ell t}, v_{rt}, v_\ell, v_{\ell r}, v_r\}$ lies on the perimeter of a trapezoid in the decomposition. We call a bag in the tree decomposition consisting of these vertices a *trapezoidal bag*. With this definition we are now ready give a proof of the fact that $\operatorname{tw}(S_n) = 4$ for all $n > 4$.

▶ **Lemma 4.** *The treewidth of S_n is equal to 4 for all $n > 4$.*

Proof. To prove the upper bound we construct a tree decomposition of S_n out of the triangular and trapezoidal bags defined above. We take the triangular bag in S_n to be the root of the tree decomposition, and recursively decompose S_n into its triangular and trapezoidal subgraphs. A bag at depth k is either a triangular or trapezoidal bag from an S_{n-k} subgraph. The children of a trapezoidal bag at depth k are the triangular bags corresponding to the two copies of S_{k-1} that make up the trapezoid. The children of a triangular bag at depth k are a trapezoidal and a triangular bag corresponding to the decomposition of S_k into a trapezoid and triangle. Every edge of S_n is contained in some triangle or trapezoid, and every triangle and trapezoid appear as a bag in the tree decomposition. For any vertex

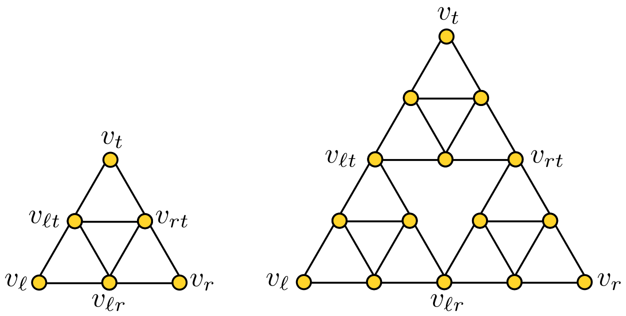


Figure 1 The Sierpínski graphs S_2 and S_3.

v in S_n if $v \in B_1, B_2$ where B_1 and B_2 are distinct bags there are two cases to consider. If B_1 is an ancestor of B_2 then v, by the construction of the bags, must be in every triangular or trapezoidal bag lying in between them. If there is no ancestry relationship, then v must lie in the intersection of the shapes defined by B_1 and B_2. Hence, there is some triangle or trapezoid containing both B_1 and B_2 which is their least common ancestor in the tree decomposition. See Figure 2 for an illustration on S_3.

To prove the lower bound it is sufficient to show that S_n contains a subdivision of the octahedron graph when $n > 4$. The octahedron graph is a forbidden minor for treewidth 3 graphs [2]. See Figure 3 for an illustration. ◀

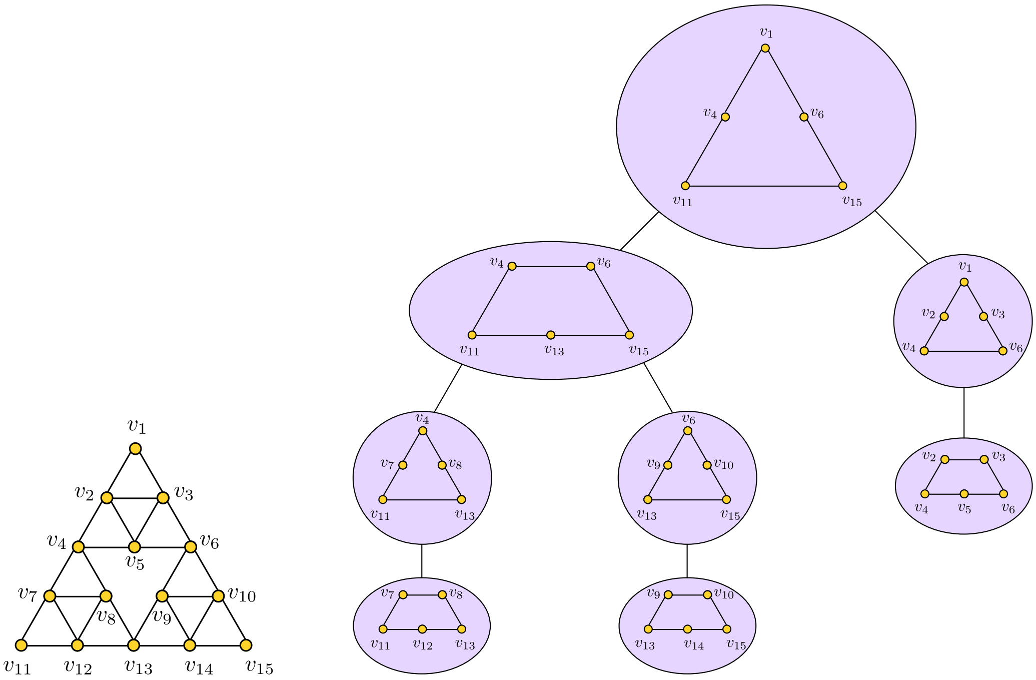


Figure 2 S_3 along with its tree decomposition.

Next we give an inductive construction of the Hanoi graph H_3^n with 3 pegs and n disks. This construction is almost identical to that of S_n, but instead of identifying vertices we connect the three copies of H_3^{n-1} with three edges. Recall that the vertices of H_3^n are configurations representing the game state, that is a vertex is an element of $\{1,2,3\}^n$. We

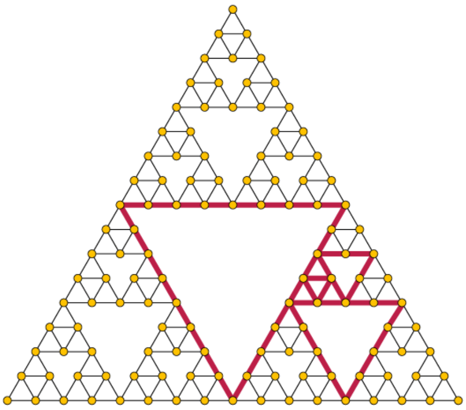


Figure 3 The graph S_5 with a subdivision of the octahedral graph highlighted in red.

define H_3^1 to be K_3 with the same planar embedding as in the case of the Sierpínski triangle and denote the vertices as the 1-tuples $(1), (2), (3)$. The cyclic ordering of the vertices does not affect our construction.

By induction we assume H_3^{n-1} has a planar embedding whose outer face is an equilateral triangle such that the corners of the triangle are the configurations corresponding with the perfect states, and we denote these vertices p_1, p_2, p_3. For $i \in \{1, 2, 3\}$ let H_i be the graph isomorphic to H_3^{n-1} with the vertex set $V(H_3^{n-1}) \times \{i\}$. We construct H_3^n out of the three H_i's and add the following edges.

1. Add an edge between $(p_1, 2)$ and $(p_1, 3)$ and denote it $e_{\ell r}$.
2. Add an edge between $(p_2, 1)$ and $(p_2, 3)$ and denote it e_{rt}.
3. Add an edge between $(p_3, 1)$ and $(p_3, 2)$ and denote it $e_{\ell t}$.

We call these three edges the *boundary edges*. The boundary edges represent the legal moves obtained by moving the largest disk. It is clear from the construction that the resulting graph embeds into the plane as an equilateral triangle with the perfect states at the corners of the triangle. See Figure 4.

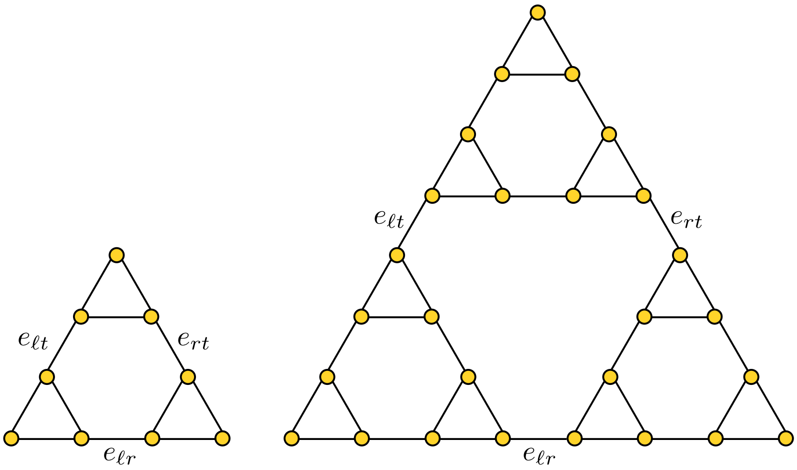


Figure 4 The Hanoi graphs H_3^2 and H_3^3. We label the boundary edges such that their index coincides with their corresponding vertex in the Sierpínski triangle.

▶ **Theorem 5.** $\mathrm{tw}(H_3^n) = 4$ *for all* $n > 4$.

Proof. To prove the lower bound we contract the boundary edges of H_3^n to create an S_n-minor. Hence, $4 = \mathrm{tw}(S_n) \leq \mathrm{tw}(H_3^n)$ for $n > 4$.

To get the inequality $\mathrm{tw}(H_3^n) \leq 4$ we inductively construct an H_3^n-minor of S_{n+1} as follows. For the base case we can easily find a copy of K_3 in S_2. Let G_1, G_2, G_3 be the three S_n subgraphs used to construct S_{n+1} and let $v_{i,j}$ be the vertex shared by G_i and G_j. By the inductive hypothesis we assume each G_i contains an H_3^{n-1}-minor which we denote by H_i. We construct an H_3^n-minor in S_{n+1} by connecting the corresponding perfect states of H_i and H_j via a path containing $v_{i,j}$ for each $i \neq j$. These paths can be chosen to be vertex-disjoint, which proves the theorem. See Figure 5 for an illustration. ◀

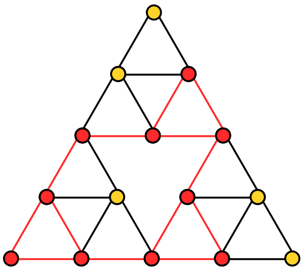


Figure 5 S_3 with an H_3^2 minor highlighted in red.

The three-peg case is simple enough that we can analyze our forbidden-state version of the puzzle directly. If two states are forbidden, the only way to separate the remaining states is to separate one recursive subgraph of the same type from the rest of the graph. In terms of the original puzzle, the two forbidden states can be described by choosing a peg and a number k and forbidding the two states where the largest k disks are on the chosen peg and the remaining $n - k$ disks are all on the same peg as each other (one of the other two pegs). The probability of a connection between two randomly-chosen states is maximized for $k = 1$, for which, for large n, the probability of a path between two randomly-chosen vertices becomes approximately $(2/3)^2 + (1/3)^2 = 5/9$. On the other hand, if three states are forbidden, it becomes possible to separate the state space into three equally-sized subgraphs by forbidding three of the six states in which the largest disk can move. For this selection, the probability of a path between two randomly-chosen vertices becomes $3(1/3)^2 = 1/3$.

4 More pegs

We conjecture that the treewidth of the Hanoi graph H_p^n is $\Theta((p-2)^n)$. By Lemma 1 the same bound would automatically apply to the recursive balanced separator orders of these graphs; by Lemma 2, this would imply an upper bound on the number of states to forbid to make the adversarial version of the Hanoi puzzle fair ($f(G)$ in Lemma 2). In this section we make progress towards this conjecture by proving the asymptotic upper bound $\mathrm{tw}(H_p^n) = O((p-2)^n)$ and the asymptotic lower bound $\mathrm{tw}(H_p^n) = \Omega(n^{-(p-1)/2} \cdot (p-2)^n)$. We obtain the lower bound by proving that every balanced separator of H_p^n (recursive or otherwise) is of this asymptotic order. This lower bound then applies to $f(G)$ in Lemma 2. Our bounds are almost tight, off only by the factor $\Theta(n^{(p-1)/2})$. We begin by proving the asymptotic upper bound, which we do by constructing a recursive balanced separator of the required order and applying Lemma 1.

▶ **Theorem 6.** *For any fixed* $p \geq 3$ *and* $n \geq 1$, $\mathrm{tw}(H_p^n) = O((p-2)^n)$.

Proof. We can recursively decompose H_p^n into p vertex-disjoint copies of H_p^{n-1} by considering the subgraphs induced by fixing the position of the largest disk in the configurations. We call a vertex a boundary vertex if in its configuration there is at least one peg occupied by no disks and the largest disk shares its peg with no other disks. These are the configurations in which the largest disk is free to move. The endpoints of edges between the H_p^{n-1} subgraphs in our decomposition are the boundary vertices.

We compute the order of our recursive balanced separator by counting the number of boundary vertices. This is the number of ways to distribute $n-1$ disks across $p-2$ pegs, hence the size of the separator is $\binom{p}{2}(p-2)^{n-1}$. Our choice of separator splits H_p^n into p subgraphs of size $\frac{1}{p}|V(H_p^n)|$. By grouping the H_p^{n-1} subgraphs into two vertex sets, we obtain a c-separator where $c \in \{\frac{\lceil p/2 \rceil}{p}, \frac{\lceil p/2 \rceil + 1}{p}, \ldots, \frac{p-1}{p}\}$ depending on our choice of vertex sets. Each subgraph can then be recursively decomposed in a similar way, and the number of vertices required in each recursive decomposition at level i is equal to $\binom{p}{2}(p-2)^{n-i}$. The theorem follows directly from Lemma 1. ◀

To prove the asymptotic lower bound we construct a new graph related to H_p^n whose treewidth is easier to compute. We can specify the positions of a subset of disks in a Hanoi puzzle by a mapping $\rho\colon [n] \to [p] \cup \{\infty\}$, where a finite value of $\rho(i)$ specifies the peg containing disk d_i and an infinite value means that disk d_i is allowed to be placed on any peg that does not also contain a specified disk. We define the *pegset* induced by ρ to be the states consistent with this specification. More formally, a vertex $v = (p_1, p_2, \ldots, p_n)$ is in the pegset induced by ρ if and only if :

1. for all $k \in [n]$, if $\rho(k) \neq \infty$ then $\rho(k) = p_k$, and
2. for all $k, l \in [n]$, if $\rho(k) = \infty \neq \rho(l)$, then $p_k \neq p_l$.

If $\rho(k) \neq \infty$ we call d_k *frozen* by ρ; further, if a peg p_k is in the image of ρ we call p_k frozen by ρ as well. Intuitively, a pegset is the result of freezing a set of disks onto a set of pegs and playing a Hanoi puzzle using only the remaining unfrozen disks and pegs. We are interested in pegsets that meet two additional properties:

3. exactly $p-3$ elements of $[p]$ have a non-empty inverse under ρ, and
4. for $j \in [p]$ either $|\rho^{-1}(j)| = \lfloor \frac{n-1}{p-2} \rfloor$ or $|\rho^{-1}(j)| = 0$.

We call such pegsets *regular pegsets*. Note that, because we still have three pegs unfrozen, and because the three-peg Hanoi graphs are connected, each regular pegset describes a connected subgraph of the Hanoi graph.

To make our analysis cleaner we assume that $n \equiv 1 \mod (p-2)$, hence properties 3 and 4 imply that there are precisely $\frac{n-1}{p-2} + 1$ unfrozen disks in a regular pegset. Note that this restriction on n does not change the overall asymptotic analysis for other values of n, as we can still lower-bound the treewidth for other n by rounding n down to a value with this restricted form.

Let I_p^n denote the graph whose vertices are the regular pegsets of H_p^n where two vertices share an edge if and only if the intersection of their corresponding pegsets is non-empty. We call I_p^n the *pegset intersection graph* of H_p^n. We characterize the adjacency condition in terms of frozen disks and pegs in Lemma 7.

▶ **Lemma 7.** *Two regular pegsets u and v are adjacent in I_p^n if and only if the following criteria are satisfied:*

1. *if a disk is frozen by both u and v, then both u and v freeze it to the same peg,*
2. *u and v each freeze exactly one peg unfrozen by the other,*
3. *if a disk is frozen by u but not by v, then the peg it is frozen on is not frozen by v, and*
4. *if a disk is frozen by v but not by u, then the peg it is frozen on is not frozen by u.*

Proof. If u and v are adjacent in I_p^n then there exists some vertex $w = (w_1, w_2, \dots, w_n)$ contained in both pegsets. By the definition of a regular pegset both u and v freeze $\frac{n-1}{p-2}$ disks evenly across $p-3$ pegs and leave $\frac{n-1}{p-2}+1$ disks unfrozen. We will now show that each of the four claims follows from the adjacency of u and v.

1. Suppose for a contradiction that a disk d_i is frozen to different pegs by u and v; then no configuration in u can equal a configuration in v since they differ at the ith component.

2. If u and v freeze the same set of pegs then a configuration in u cannot equal a configuration in v since they will differ on the components corresponding to frozen disks. Now, assume u freezes more than one peg left unfrozen by v. A vertex w in the intersection of u and v would have a configuration that matches both u and v on their frozen disks, but the total number of disks frozen by u and v is at least $(p-1)\cdot\frac{n-1}{p-2}$; then w has more than n disks, contradicting the fact that w is a valid configuration.

3. Let u freeze the disk d_i onto the peg p_k and assume v does not freeze d_i. If v also freezes p_k then v must freeze $\frac{n-1}{p-2}$ disks onto p_k while leaving d_i unfrozen, hence there is no configuration in v that places d_i onto p_k.

4. Identical to 3.

We are now ready to prove the converse. Let u and v be pegsets in I_p^n such that conditions 1 through 4 hold. Conditions 1 and 2 tell us that the configurations of u and v coincide with one another for $(p-4)\cdot\frac{n-1}{p-2}$ disks evenly distributed across $p-4$ pegs. The remaining $2\cdot\frac{n-1}{p-2}$ disks are left unfrozen by either u or v; call the set of these disks U. Conditions 3 and 4 ensure that we can choose a peg that is frozen by either u or v, but not both, and place $\frac{n-1}{p-2}$ of the disks in U onto this peg which yields a configuration shared by both u and v. ◀

As a consequence of Lemma 7 we can describe how to traverse an edge from a pegset u to a pegset v in I_p^n by freezing and unfreezing disks. We place $\frac{n-1}{p-2}$ of the disks left unfrozen by u onto the peg frozen by v but left unfrozen by u. Then, we take the peg frozen by u and left unfrozen by v and unfreeze every disk on it.

The asymptotic lower bound on $\mathrm{tw}(H_p^n)$ will be derived from an asymptotic lower bound on $\mathrm{tw}(I_p^n)$. To compute the treewidth of I_p^n we first need to prove that it is vertex-transitive and compute its diameter.

▶ **Lemma 8.** *I_p^n is vertex-transitive.*

Proof. We define a family of automorphisms $\phi_{i,j}$ which swap the roles of d_i and d_j in some pegset. The lemma follows from the fact that we can transform a pegset u to any other pegset v by a sequence of swap operations. For any pegset u we define the image of u under $\phi_{i,j}$ to be

$$\phi_{i,j}(u)(k) = \begin{cases} u(i) & k = j \\ u(j) & k = i \\ u(k) & \text{otherwise} \end{cases} .$$

Let u and v be adjacent pegsets in I_p^n. By U_u and U_v we denote the sets of disks left unfrozen by u and v, respectively. By Lemma 7 there are pegs p_u and p_v such that u freezes disks onto p_u but not p_v, and v freezes disk onto p_v but not p_u. Further, traversing the edge from u to v is equivalent to placing $\frac{n-1}{p-2}$ disks from U_u onto p_k and treating the disks frozen to p_u as unfrozen. If u and v are adjacent then $\phi_{i,j}(u)$ and $\phi_{i,j}(v)$ are also adjacent, since swapping the labels of two disks does not affect the traversal process. If $\phi_{i,j}(u)$ and $\phi_{i,j}(v)$ are adjacent then so are u and v, by the above and the fact that $\phi_{i,j} = \phi_{i,j}^{-1}$. ◀

▶ **Lemma 9.** *The diameter of I_p^n is $\Theta(n)$.*

Proof. Let u and v be pegsets in I_p^n. Let $k = \frac{n-1}{p-2}$. If u and v do not freeze the same set of pegs, we can, by freezing and unfreezing disks, walk along a path in I_p^n of length depending only on p, to a configuration that does freeze the same set of pegs as v, and continue with the process below. Therefore, assume u and v do freeze the same set of pegs, and label this set of pegs in increasing order by index as $Q = \{q_1, q_2, \ldots, q_{p-3}\}$. For $i = 1, \ldots, p-3$, let U_i be the set of disks frozen on q_i by u, and let V_i be the set frozen on q_i by v.

For all $i = 1, \ldots, p-3$, we iteratively transform u into v, one peg at a time. For a given peg q_i, the process for transforming U_i into V_i is as follows. There are three cases:

1. There exists some disk $d \in V_i \setminus U_i$ that is unfrozen by u,
2. There exists some disk $d \in V_i \setminus U_i$ that is frozen on some other peg by u,
3. or $U_i = V_i$.

In case 1, unfreeze q_i, then freeze an arbitrary peg $q_l \notin Q$, to obtain a new pegset w adjacent to the current pegset. Since each pegset leaves $\frac{n-1}{p-2} + 1$ disks unfrozen, let the new pegset freeze onto q_l all but one of the disks unfrozen by u. Let the omitted disk, d, be one in V_i that is unfrozen by u. Choose some $d' \in U_i \setminus V_i$ (one exists since $|U_i| = |V_i|$), then unfreeze q_l; freeze onto q_i the set $(U_i \setminus \{d'\}) \cup \{d\}$, to obtain a new adjacent pegset where d' is replaced by d.

Repeat this process until case 1 no longer applies, i.e. until every remaining $d \in V_i \setminus U_i$ is frozen by u. Then (case 2) consider some such d. u does not freeze d on a peg q_r to which this process has already been applied, since all such pegs now agree with v. Therefore, u freezes d on some peg q_s to which the process has not yet been applied. Unfreeze q_s and freeze an arbitrary unfrozen peg q_l to obtain the next pegset in the process; when doing so, some unfrozen disk d'' remains unfrozen. Then again freeze q_s, but omit d and instead freeze d'' onto q_s. d is now unfrozen, and we proceed as in case 1. Repeat case 2 until case 3 applies.

Repeating the overall process for every peg gives a path from u to v of length $O(n)$. ◀

Lemmas 8 and 9 allow us to apply the following lemma due to Babai and Szegedy to obtain a lower bound of $\Omega(\frac{1}{n}V(|I_p^n|))$ on the *vertex expansion* of I_p^n.

▶ **Definition 10.** *The vertex expansion of a graph G is equal to*

$$\min_{S \subseteq V(G): 1 \leq |S| \leq \frac{1}{2}} \frac{|\partial S|}{|S|},$$

where ∂S is the union of the neighborhoods, in $G \setminus S$, of vertices in S.

▶ **Lemma 11** (Babai and Szegedy [3]). *Let G be a vertex-transitive graph. Then the vertex expansion of G is $\Omega(1/d)$, where d is the diameter of G.*

▶ **Lemma 12.** *The treewidth of I_p^n is $\Omega(\frac{1}{n}|V(I_p^n)|)$.*

Proof. By applying Lemmas 8, 9, and 11, along with the definition of vertex expansion, we have $|\partial S| = \Omega(\frac{|S|}{n})$ for all $S \subseteq V(I_p^n)$ with $0 \leq |S| \leq \frac{|V(I_p^n)|}{2}$, which implies that the size of any balanced vertex separator of I_p^n is bounded from below by $\Omega(\frac{|V(I_p^n)|}{n})$. It follows that the treewidth of I_p^n is also bounded from below by $\Omega(\frac{|V(I_p^n)|}{n})$. ◀

We now count the number of pegsets in $V(I_p^n)$.

▶ **Lemma 13.** *The number of regular pegsets in I_p^n is $\Theta(n^{-(p-3)/2} \cdot (p-2)^n)$.*

Proof. There are $\binom{p}{p-3}$ ways to choose the frozen pegs. Each pegset divides the disks into $p-2$ sets of (almost) equal size and there are $\frac{n!}{(\frac{n}{p-2})!)^{p-2}}$ ways to choose the sets. This is because there are $n!$ ways to order the disks, but we only care about the ordering of the $p-2$ partitions of the disks, hence we divide by $\left(\frac{n}{p-2}!\right)^{p-2}$. (Asymptotically, we may assume $n \equiv 0 \mod p-2$.) In total there are $\binom{p}{p-3} \cdot \frac{n!}{((\frac{n}{p-2})!)^{p-2}}$ pegsets. Since p is fixed, we apply Stirling's approximation to $\frac{n!}{((\frac{n}{p-2})!)^{p-2}}$ to obtain the result. ◀

By applying Lemmas 12 and 13 we obtain the following corollary.

▶ **Corollary 14.** $\operatorname{tw}(I_p^n) = \Omega(n^{-(p-1)/2} \cdot (p-2)^n)$.

Next we show how to obtain a lower bound of $\operatorname{tw}(H_p^n)$ from Corollary 14. Since we have a lower bound on the treewidth of I_p^n, Lemma 3 guarantees the existence of a haven of a useful order. The idea behind Lemma 15 is to take a haven of order $\Omega(n^{-(p-1)/2} \cdot (p-2)^n)$ in I_p^n and modify it to create a haven of the same order in H_p^n.

▶ **Lemma 15.** $\operatorname{tw}(H_p^n) = \Omega(\operatorname{tw}(I_p^n))$.

Proof. Let $k = \operatorname{tw}(I_p^n) + 1 = \Omega(n^{-(p-1)/2} \cdot (p-2)^n)$. By Lemma 3, I_p^n has a haven of order k. Call this haven ϕ. Recall that a haven describes an evasion strategy for a robber in a cops-and-robbers game. Intuitively, if a robber can evade the cops in I_p^n, the same robber can also evade the cops in H_p^n by playing only on states that belong to pegsets of I_p^n and by paying attention only to which of those pegsets are occupied by at least one cop. We formalize this strategy below by constructing a haven for H_p^n from ϕ. Because a cop moving in H_p^n may simultaneously occupy a constant number of pegsets in I_p^n, the order of the haven we construct is a constant factor smaller than that of ϕ.

Every vertex in I_p^n corresponds to a pegset; every pegset corresponds to a set of configurations in the Towers of Hanoi game. Each of these configurations corresponds to a vertex in H_p^n. Define the function $f : \mathcal{P}(V(H_p^n)) \to \mathcal{P}(V(I_p^n))$, where for $X \subseteq V(H_p^n)$, $f(X)$ is the set of vertices in I_p^n whose corresponding pegsets contain configurations in X. Define $g : \mathcal{P}(V(I_p^n)) \to \mathcal{P}(V(H_p^n))$, such that for $X' \subseteq V(I_p^n)$, $g(X')$ is the set of all configurations belonging to pegsets in X'. Let $g(X') = \emptyset$ if $X' = \emptyset$.

Define $\psi : \mathcal{P}(V(H_p^n)) \to \mathcal{P}(V(H_p^n))$, such that $\psi(X)$ is the connected component containing $g(\phi(f(X)))$. To show that ψ is a haven, it suffices to show that:

1. for all $X \subseteq V(H_p^n)$, $\psi(X)$ is well-defined—i.e. $g(\phi(f(X)))$ is connected and nonempty whenever $\phi(f(X))$ is nonempty,
2. for $Z \subseteq V(H_p^n)$, $\psi(Z) \subseteq \psi(X)$ whenever $X \subseteq Z$, and
3. $|X| = \Omega(f(X))$.

For (1), to see that $g(\phi(f(X)))$ is connected in $H_p^n \setminus X$, consider any pair of configurations $u, v \in g(\phi(f(X)))$. u and v belong to pegsets a and b (respectively) in $\phi(f(X))$. a has a path P to b in $\phi(f(X))$, since $\phi(f(X))$ is connected. Every vertex (pegset) w in this path corresponds to the set $W' = g(w) \subseteq g(\phi(f(X)))$ of all configurations belonging to the pegset w. $W' \cap X = \emptyset$, or else by the definition of f, w would be in $f(X)$, contradicting the fact that $w \in \phi(f(X))$. Furthermore, W' is connected, since it is isomorphic to H_3^d (where $d < n$). Also, every edge (w_1, w_2) in P corresponds to a vertex $w' \in g(\phi(f(X)))$ belonging to $W_1' = g(w_1)$ and $W_2' = g(w_2)$. Therefore, u has a path to v in $H_p^n \setminus X$, obtained by traversing an H_3^d copy W' for every vertex $w \in P$, and moving between H_3^d copies W' and W'' that intersect at a vertex in $g(\phi(f(X)))$ for every edge in P.

For (2), if $X \subseteq Z \subseteq V(H_p^n)$, then $f(X) \subseteq f(Z)$. Since ϕ is a haven, $\phi(f(Z)) \subseteq \phi(f(X))$. Therefore, $g(\phi(f(Z))) \subseteq g(\phi(f(X)))$, and both $g(\phi(f(Z)))$ and $g(\phi(f(X)))$ are connected. If $\phi(f(Z)) = \emptyset$, then $\psi(X) = \emptyset$, and (2) is true. Therefore, suppose $\phi(f(Z)) \neq \emptyset$. Suppose for a contradiction that $\psi(Z) \not\subseteq \psi(X)$. Let u be a vertex in $\psi(Z) \cap \psi(X)$ (this intersection is nontrivial since it includes $g(\phi(f(Z)))$), and let v be a vertex in $\psi(Z) \setminus \psi(X)$. Suppose $(u, v) \in E(H_p^n)$. (Such a pair must exist because $\psi(Z)$ is connected.) Since $X \subseteq Z$,

$$u, v \in \psi(Z) \subseteq V(H_p^n) \setminus Z \subseteq V(H_p^n) \setminus X.$$

However, since $v \notin \psi(X)$, this contradicts that $\psi(X)$ is a connected component in $H_p^n \setminus X$.

(3) follows from the fact that $|f(X)| \leq (p-2)|X|$, since every vertex belongs to at most $p-2$ regular pegsets. ◀

We have now proven the second theorem of the section.

▶ **Theorem 16.** *For any fixed $p \geq 4$, $\mathrm{tw}(H_p^n) = \Omega(n^{-(p-1)/2} \cdot (p-2)^n)$.*

In Appendix A we prove a lower bound on the treewidth of four-peg Hanoi graphs that, while still separated by a polynomial factor from the upper bound, is tighter than the one above.

5 Conclusion

Theorem 16 and Theorem 17, together with Theorem 5 and Theorem 6, give nearly tight asymptotic bounds on the number of states, in the adversarial version of the Towers of Hanoi game we proposed in the introduction, that the first player must forbid in order to ensure better than even odds of defeating the second player. This raises additional questions. First, suppose the first player forbids enough states to separate the graph in a balanced way, but the second player is fortunate enough to have starting and ending positions in the same connected component. What is the optimal strategy for the second player, and how many moves will this strategy take? Must this strategy be formulated in graph-theoretic terms, or is there an algorithm that consists of moving the disks in an intuitive way?

Theorem 17 (in Appendix A) improves the lower bound of Theorem 16 when $p = 4$; one question would be to see whether the technique in the proof of Theorem 17 could be adapted to deal more generally with the structure of pegset intersection graphs when $p \geq 4$, yielding a bound of $\Omega(\frac{(p-2)^n}{n})$ in general when $p \geq 4$. However, this still would not eliminate the asymptotic gap between our upper and lower bounds.

To this end, Corollary 25 gives a lower bound on the treewidth of the Kneser graph that is new when $2k+1 \leq n \leq 3k-1$. Can this lower bound be tightened? Harvey and Wood [12] showed that in this case (when $2k+1 \leq n \leq 3k-1$),

$$\mathrm{tw}(\mathrm{Kn}(n,k)) < \binom{n-1}{k} - 1.$$

Since $\binom{n-1}{k} = \Theta(\binom{n}{k})$ when $2k+1 \leq n \leq 3k-1$, this upper bound does not imply sublinear treewidth. However, if in fact the treewidth is sublinear, and can be used to obtain a sublinear vertex separator in $\mathrm{Ds}(n)$ (defined in Appendix A), then combined with a proof of asymptotic tightness in Lemma 31 and Lemma 35, this would imply that $\mathrm{tw}(H_4^n)$ is $o(2^n)$, proving that the upper bound in Theorem 6 is not tight. This would be surprising, as the family of separators given in Theorem 6 seems intuitively to target the “weakest” parts of the graph.

Another possible line of further research is whether the bound given in Lemma 33 is tight for the tensor product, and what can be said about other graph products. As stated in the introduction, Kozawa et al. [14] gave lower bounds for the Cartesian and strong products. Since the strong product of a graph has the same vertices as and a superset of the edges of the tensor product, our lower bound in Lemma 33 for the tensor product's treewidth immediately gives a lower bound on the treewidth of the strong product. However, Kozawa et al. [14] gave a stronger lower bound for the strong product. One question would be whether a comparable improvement over our bound can be proven for the tensor product.

References

1 Danielle Arett and Suzanne Dorée. Coloring and counting on the tower of Hanoi graphs. *Mathematics Magazine*, 83(3):200–209, 2010. doi:10.4169/002557010x494841.

2 Stefan Arnborg, Andrzej Proskurowski, and Derek G. Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80(1):1–19, 1990. doi:10.1016/0012-365X(90)90292-P.

3 László Babai and Mario Szegedy. Local expansion of symmetrical graphs. *Combinatorics, Probability and Computing*, 1(1):1–11, 1992. doi:10.1017/S0963548300000031.

4 Hans L. Bodlaender. Fixed-parameter tractability of treewidth and pathwidth. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *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 196–227. Springer, 2012. doi:10.1007/978-3-642-30891-8_12.

5 Thierry Bousch. La quatrième tour de Hanoï. *Bulletin of the Belgian Mathematical Society – Simon Stevin*, 21(5):895–912, 2014. URL: http://projecteuclid.org/euclid.bbms/1420071861.

6 Boštjan Brešar and Simon Špacapan. On the connectivity of the direct product of graphs. *The Australasian Journal of Combinatorics*, 41:45–56, 2008. URL: https://ajc.maths.uq.edu.au/pdf/41/ajc_v41_p045.pdf.

7 David Eppstein. Treewidth of deep Sierpiński sieve graph. Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/36542.

8 David Eppstein and Elham Havvaei. Parameterized leaf power recognition via embedding into graph products. In Christophe Paul and Michal Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2018.16.

9 Jeff Erickson. Computational topology: Treewidth. Lecture Notes, 2009. URL: http://jeffe.cs.illinois.edu/teaching/comptop/2009/notes/treewidth.pdf.

10 Andreas M. Hinz et al. *The Tower of Hanoi —Myths and Maths.* Birkhäuser Basel, 2013.

11 Peter Frankl. A new short proof for the Kruskal–Katona theorem. *Discrete Mathematics*, 48(2–3):327–329, 1984. doi:10.1016/0012-365X(84)90193-6.

12 Daniel J. Harvey and David R. Wood. Treewidth of the Kneser Graph and the Erdős–Ko–Rado theorem. *Electronic Journal of Combinatorics*, 21(1):P1.48, 2014. doi:10.37236/3971.

13 Wilfried Imrich, Sandi Klavžar, and Douglas F. Rall. *Topics in Graph Theory: Graphs and Their Cartesian Product.* A K Peters, 2008.

14 Kyohei Kozawa, Yota Otachi, and Koichi Yamazaki. Lower bounds for treewidth of product graphs. *Discrete Applied Mathematics*, 162(C):251–258, January 2014.

15 L. Lovász. Kneser's conjecture, chromatic number, and homotopy. *Journal of Combinatorial Theory, Series A*, 25(3):319–324, 1978. doi:10.1016/0097-3165(78)90022-5.

16 L. Lovász. *Combinatorial Problems and Exercises.* AMS/Chelsea publication. North-Holland Publishing Company, 1993.

17 J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

18 Miodrag S. Petković. *Famous Puzzles of Great Mathematicians*. American Mathematical Society, 2009.

19 Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993. doi:10.1006/jctb.1993.1027.

20 B. M. Stewart and J. S. Frame. Problem 3918 and solution. *The American Mathematical Monthly*, 48(3):216–219, 1941. doi:10.2307/2304268.

21 Mario Valencia-Pabon and Juan-Carlos Vera. On the diameter of Kneser graphs. *Discrete Mathematics*, 305(1–3):383–385, 2005. doi:10.1016/j.disc.2005.10.001.

A Four pegs

Theorems 6 and 16, together, give upper and lower bounds that differ by a polynomial factor in the number of disks of the Towers of Hanoi puzzle. Compared to the overall exponential size of the bound, this is a small gap, and it is tempting to try to close it further. The proof of Theorem 16 identifies the pegset intersection graph (I_p^n) as a hard part of the graph to separate, and leverages the vertex-transitive structure of this graph.

However, there are many configurations in the game that are ignored by focusing on the I_p^n graph: namely, all configurations where the numbers of disks on the pegs are arbitrary, i.e., not constrained to be equal to $\lfloor \frac{n}{p-2} \rfloor$ for $p-3$ of the pegs. We broaden our analysis of pegsets to prove the main result of this section:

▶ **Theorem 17.** $\mathrm{tw}(H_4^n) = \Omega(\frac{2^n}{n})$.

We begin by generalizing the pegset intersection graph beyond regular pegsets.

▶ **Definition 18.** *Let G_4^n be a graph whose vertices are the pegsets of H_4^n that freeze only one peg and that freeze at most $\lfloor \frac{n-1}{2} \rfloor$ disks onto that peg. In this graph, let vertices u and v be adjacent whenever the pegsets u and v freeze mutually disjoint sets of disks, and freeze them onto separate pegs.*

Clearly I_4^n is an induced subgraph of G_4^n. We prove our improved bound by analyzing the relationship between G_4^n and the *Kneser graph*.

▶ **Definition 19** (Lovasz [15]). *Let $[n] = \{1, \ldots, n\}$ be an indexing of the objects in an arbitrary set. The Kneser graph, denoted $\mathrm{Kn}(n,k)$, is the graph whose vertices correspond to the k-element subsets of $[n]$, and whose edges are the pairs of vertices whose corresponding subsets are disjoint.*

We restrict our attention to Kneser graphs that are connected, namely the graphs $\mathrm{Kn}(n,k)$ where $n \geq 2k+1$.

The condition on disjoint subsets in the definition of Kneser graphs is analogous to the condition on disjoint subsets of pegs in the definition of G_4^n. (In fact, for any given $k \leq \lfloor \frac{n-1}{2} \rfloor$, the pegsets that freeze exactly k disks induce as a subgraph of G_4^n the tensor product of $\mathrm{Kn}(n,k)$ with a 4-clique—see Definition 32.) However, G_4^n also includes a separate condition, of having different frozen pegs. An additional complication is that G_4^n allows sets of different sizes rather than only considering sets of a single size k. To account for all set sizes appropriately, we introduce a generalization of the Kneser graph:

▶ **Definition 20.** *Let the disjoint subset graph, denoted $\mathrm{Ds}(n,r)$, be the graph whose vertices are identified with the subsets $s \subseteq [n]$ with $|s| \leq r$, and whose edges are the pairs of vertices whose corresponding subsets are disjoint.*

For convenience, we let $\mathrm{Ds}(n) = \mathrm{Ds}(n, \frac{n-1}{2})$. Clearly $|V(\mathrm{Ds}(n))| \approx 2^{n-1}$. Then $V(G_4^n)$ consists of four copies of $V(\mathrm{Ds}(n))$, with pegsets u and v connected iff they are in different copies and they share an edge in $\mathrm{Ds}(n)$. In Lemma 21 we bound the treewidth of $\mathrm{Ds}(n)$, after which we will use the relationship between G_4^n and $\mathrm{Ds}(n)$ to prove Theorem 17.

▶ **Lemma 21.** $\mathrm{tw}(\mathrm{Ds}(n)) = \Omega(\frac{2^n}{n})$.

We defer the formal proof of Lemma 21 to later but outline a proof sketch below. The idea of the proof is to observe that $\mathrm{Ds}(n)$ consists of $\frac{n-1}{2}$ Kneser graph "slices." We make observations analogous to those leading to Corollary 14: Kneser graphs are vertex-transitive (Remark 24) and have diameter $O(n)$ (Lemma 22), implying that for all $0 \leq k \leq \frac{n-1}{2}$, $\mathrm{tw}(\mathrm{Kn}(n,k)) = \Omega(\frac{1}{n}|V(\mathrm{Kn}(n,k))|)$ (Corollary 25). Since

$$|V(\mathrm{Ds}(n))| = \sum_{k=0}^{\frac{n-1}{2}} |V(\mathrm{Kn}(n,k))|,$$

Lemma 21 then follows if we can, intuitively, show that the Kneser slices are hard to separate from one another. We formalize this notion and show that it is true for most of the slices. The argument relies on the subset definitions of the Kneser graphs' vertices, and makes use of the *Kruskal–Katona Theorem* (Corollary 29).

We prove that given a balanced vertex separator X for $\mathrm{Ds}(n)$, either:

1. X contains a large number of the vertices in $\mathrm{Ds}(n)$ (at least an $\Omega(\frac{1}{n})$ factor), or
2. after removing X from $\mathrm{Ds}(n)$, there is still a large connected component in $\mathrm{Ds}(n)$, leading to a contradiction.

In the second case, we derive the contradiction as follows: we observe that after removing X from $\mathrm{Ds}(n)$, if case (1) does not hold, then most of the vertices of $\mathrm{Ds}(n)$ lie in Kneser slices that have large connected components, since their intersection with X contains too few vertices for a balanced separator. Call this set of Kneser slices $K_{conn}(X)$. We prove that every pair of subgraphs $G_k = \mathrm{Kn}(n,k)$ and $G_l = \mathrm{Kn}(n,l)$ in $K_{conn}(X)$ have large connected components A_k and A_l that share an edge. Therefore, these large connected components, together, form a large connected component in $\mathrm{Ds}(n) \setminus X$, from which we derive the desired contradiction.

Finally, we use our lower bound on the treewidth of $\mathrm{Ds}(n)$ to derive a lower bound on the treewidth of G_4^n, and in turn on the treewidth of H_4^n. We obtain the former by proving a more general claim about the treewidth of the tensor product of two graphs, and the latter by a proof analogous to that of Lemma 15.

We begin by showing the required lower bound on the treewidth of the Kneser graph. We use the following result of Valencia-Pabon and Vera:

▶ **Lemma 22** (Valencia-Pabon and Vera [21]). *If* $1 \leq k \leq \lfloor \frac{n-1}{2} \rfloor$, *then the diameter of* $\mathrm{Kn}(n,k)$ *is* $\lceil \frac{k-1}{n-2k} \rceil + 1$.

▶ Remark 23. When $k \leq \frac{n-1}{2}$, the diameter in Lemma 22 is $O(n)$.

The following fact about Kneser graphs is well known; it also follows from a straightforward adaptation of the proof of Lemma 8.

▶ Remark 24. All Kneser graphs are vertex-transitive.

Combining Lemmas 11 and 22 with Remarks 23 and 24, and observing the relationship between vertex expansion, balanced separators, and treewidth (as we did in the proof of Lemma 12), gives the following corollary:

▶ **Corollary 25.** *For all k such that $1 \leq k \leq \frac{n-1}{2}$, $\mathrm{tw}(\mathrm{Kn}(n,k)) = \Omega(\frac{1}{n}|V(\mathrm{Kn}(n,k))|)$, and for every constant c, the minimum size of a c-separator in $\mathrm{Kn}(n,k)$ is $\Omega(\frac{1}{n}|V(\mathrm{Kn}(n,k))|)$.*

Before turning to the interfaces between the Kneser slices, we establish a threshold value such that most of the vertices of $\mathrm{Ds}(n)$ lie in $\mathrm{Kn}(n,k)$ slices with values of k exceeding this threshold. Restricting our attention (in Lemma 27) to these slices will allow us to prove the mutual connectedness of the large connected components in case (2).

▶ **Lemma 26.** *For every constant β with $\frac{1}{2} < \beta < 1$, there exists a constant ε such that*

$$\lim_{n\to\infty} \frac{\sum_{k=\frac{n}{2}-\varepsilon\sqrt{n}}^{\frac{n}{2}} |V(\mathrm{Kn}(n,k))|}{|V(\mathrm{Ds}(n))|} \geq \beta.$$

Proof. Let $B(n,p)$ denote the binomial distribution parameterized with probability p. The standard deviation of $B(n,\frac{1}{2})$ is $\frac{\sqrt{n}}{2}$. If f is the probability mass function of $B(n,\frac{1}{2})$, then $f(k) = \frac{1}{2^n}\binom{n}{k}$.

Let X be a random variable distributed according to $B(n,p)$.

By Chebyshev's inequality,

$$Pr[|X - \frac{n}{2}| \geq \varepsilon\sqrt{n}] \leq \frac{1}{4\varepsilon^2}.$$

Setting $\varepsilon = \frac{1}{2\sqrt{1-\beta}}$, so that $\beta = 1 - \frac{1}{4\varepsilon^2}$, yields the desired result, since

$$\sum_{k=\frac{n}{2}-\varepsilon\sqrt{n}}^{\frac{n}{2}} \binom{n}{k} = \frac{1}{2} \sum_{k=\frac{n}{2}-\varepsilon\sqrt{n}}^{\frac{n}{2}+\varepsilon\sqrt{n}} \binom{n}{k} = 2^{n-1} \cdot Pr[|X - \frac{n}{2}| \leq \varepsilon\sqrt{n}] \geq 2^{n-1}(1 - \frac{1}{4\varepsilon^2}),$$

and since $|V(\mathrm{Ds}(n))| = 2^{n-1}$. ◀

In Lemma 30 we will prove the existence of the large connected component from which the contradiction is derived in case (2) of the discussion following the statement of Lemma 21. To do so, we will use the following lemma:

▶ **Lemma 27.** *Let $\varepsilon > 0$ be fixed. Suppose $\frac{n-1}{2} - \varepsilon\sqrt{n} \leq l < k \leq \frac{n-1}{2}$, and let A_k and A_l be subsets, respectively, of the vertices in the $\mathrm{Kn}(n,k)$ and $\mathrm{Kn}(n,l)$ subgraphs of $\mathrm{Ds}(n)$. Suppose further that $|A_k| \geq d|V(\mathrm{Kn}(n,k))|$ and $|A_l| \geq d|V(\mathrm{Kn}(n,l))|$, where $d > \frac{1}{2}$ is a constant. Then, if n is sufficiently large, A_k and A_l share an edge.*

The proof of Lemma 27 uses the *Kruskal–Katona Theorem* (Corollary 29), which provides a lower bound, given a collection $\mathcal{F}$ of k-element subsets of $[n]$, on the number of l-element subsets of $[n]$ that are subsets of sets in $\mathcal{F}$. The following formulation of the Kruskal-Katona theorem is due to Lovász (Frankl gave a short proof):

▶ **Theorem 28** (Kruskal–Katona Theorem [11],[16])**.** *Let $\mathcal{F}$ be a family of k-element subsets of $[n]$, and let $\mathcal{E}$ be the set of all $k-1$-element subsets of sets in $\mathcal{F}$. Then whenever $|\mathcal{F}| \geq \binom{m}{k}$, $|\mathcal{E}| \geq \binom{m}{k-1}$.*

Applying induction on $l = k-1, \ldots, 1$ to Theorem 28 implies the following corollary:

▶ **Corollary 29.** *Let $\mathcal{F}$ be a family of k-element subsets of $[n]$, and let $\mathcal{E}$ be the set of all l-element subsets of sets in $\mathcal{F}$, where $1 \leq l < k$. Then whenever $|\mathcal{F}| \geq \binom{m}{k}$, $|\mathcal{E}| \geq \binom{m}{l}$.*

Using Corollary 29, we prove Lemma 27:

Proof of Lemma 27. For every $v \in V(\mathrm{Ds}(n))$, view v as the k-size subset with which it is identified, and let $\overline{v}$ be the set complement of v.

Let $B_k = \{\overline{v} \mid v \in A_k\}$. Define a function δ_l mapping vertices in $\mathrm{Kn}(n,k)$ to their neighborhoods in $\mathrm{Kn}(n,l)$: for all $v \in V(\mathrm{Kn}(n,k))$, let $\delta_l(v) = \{w \in V(\mathrm{Kn}(n,l)) \mid (v,w) \in E(\mathrm{Ds}(n))\}$.

Extend the domain of δ_l to sets of vertices in $\mathrm{Kn}(n,k)$: for all $Z \subseteq V(\mathrm{Kn}(n,k))$, let $\delta_l(Z) = \bigcup_{v \in Z} \delta_l(v)$.

Clearly, a vertex $u \in \mathrm{Kn}(n,l)$ is in $\delta_l(A_k)$ iff there exists some $w \in B_k$ such that, viewing the vertices in their combinatorial sense, $u \subseteq w$.

I.e., $\delta_l(A_k)$ consists precisely of the vertices that are identified with subsets of vertices in B_k. Since

$$|B_k| = |A_k| > \frac{1}{2}|\mathrm{Kn}(n,k)| = \frac{1}{2}\binom{n}{k} \geq \binom{n-1}{n-k},$$

Corollary 29 implies that

$$|\delta_l(A_k)| \geq \binom{n-1}{l} \geq (\frac{1}{2} - o(1))\binom{n}{l} - 1 \geq (\frac{1}{2} - o(1))|V(\mathrm{Kn}(n,l))| - 1.$$

In the above inequalities we use the (easily verified) fact that whenever $\frac{n-1}{2} - \varepsilon\sqrt{n} \leq i \leq \frac{n-1}{2}$,

$$\binom{n-1}{i} \geq (\frac{1}{2} - o(1))\binom{n}{i} - 1,$$

and

$$\binom{n-1}{n-i} \leq \frac{1}{2}\binom{n}{i}.$$

Since by assumption $|A_l| \geq d|V(\mathrm{Kn}(n,l))|$ with $d > \frac{1}{2}$, this implies that for sufficiently large n, $\delta_l(A_k) \cap A_l \neq \emptyset$. That is, some vertex in A_k shares an edge with some vertex in A_l. ◀

We are ready to formalize case (2) (Lemma 30) in the discussion following the statement of Lemma 21.

▶ **Lemma 30.** *Let X be a vertex separator for $\mathrm{Ds}(n)$. Let $\frac{1}{2} < c < 1$ and $\varepsilon > 0$ be constants. Let*

$$K_{big} = \{\mathrm{Kn}(n,k) | \frac{n-1}{2} - \varepsilon\sqrt{n} \leq k \leq \frac{n-1}{2}\}$$

be the largest $\varepsilon\sqrt{n}$ Kneser subgraphs of $\mathrm{Ds}(n)$. Let

$$K_{conn}(X) = \{\mathrm{Kn}(n,k) \in K_{big} | \frac{|X \cap V(\mathrm{Kn}(n,k))|}{|V(\mathrm{Kn}(n,k))|} < f(n)\},$$

where f is any function such that $f(n) = O(\frac{1}{n})$.

Then if n is sufficiently large, for all $\mathrm{Kn}(n,k) \in K_{conn}(X)$, $\mathrm{Kn}(n,k) \setminus X$ has a connected component A_k of size at least $c(1 - O(\frac{1}{n}))|V(\mathrm{Kn}(n,k))|$, and for all $l \neq k$, if $\mathrm{Kn}(n,l) \in K_{conn}(X)$, then A_k and A_l share an edge.

Proof. By Corollary 25, for all $\mathrm{Kn}(n,k) \in K_{conn}(X)$, the minimum c-separator size for $\mathrm{Kn}(n,k)$ is $\Omega(\frac{1}{n}|V(\mathrm{Kn}(n,k))|)$, which by assumption is more than the vertices of X that lie in $\mathrm{Kn}(n,k)$ —at least when n is sufficiently large. This implies that A_k is of the stated size. For the second part of the claim, consider any A_k, A_l pair. A_k and A_l are connected by Lemma 27, since $c(1-O(\frac{1}{n})) \geq d$ for every constant d such that $c \geq d > \frac{1}{2}$. The lemma follows. ◀

We are now ready to prove Lemma 21. We choose numerical values instead of symbols for some of the constants that appear in the proof to make the argument more intuitive, although there are other values that work.

Proof of Lemma 21. Choose any constant $\frac{1}{2} < c < \frac{4}{7}$. Let X be a c-separator for $\mathrm{Ds}(n)$.

We will show that either X contains many vertices from large Kneser slices (those in $K_{sep}(X)$, which we define below), or most (more than a factor of c) of the vertices of $\mathrm{Ds}(n) \setminus X$ lie in a large connected component, so that X is not a c-separator.

Let K_{big} be the set of $\mathrm{Kn}(n,k)$ subgraphs with $\frac{n-1}{2} - \varepsilon\sqrt{n} \leq k \leq \frac{n-1}{2}$, where ε is chosen so that $\frac{|V(K_{big})|}{|V(\mathrm{Ds}(n))|} \geq \frac{8}{9}$. (We choose $\frac{8}{9}$ to make the argument work for $c < \frac{4}{7}$.) Let

$$K_{sep}(X) = \{\mathrm{Kn}(n,k) \in K_{big} | \frac{|X \cap \mathrm{Kn}(n,k)|}{|\,\mathrm{Kn}(n,k)|} \geq f(n)\},$$

where $f(n) = \Theta(\frac{1}{n})$ is the lower bound given by Corollary 25 on the minimum $\frac{5}{7}$-separator size for $\mathrm{Kn}(n,k) \in K_{big}$. (We choose $\frac{5}{7}$ because it produces the desired result for $c < \frac{4}{7}$.)

Let $K_{conn}(X) = K_{big} \setminus K_{sep}(X)$. There are two cases:

1. $\frac{|V(K_{sep}(X))|}{|V(K_{big})|} \geq \frac{1}{10}$.
2. $\frac{|V(K_{conn}(X))|}{|V(K_{big})|} > \frac{9}{10}$.

(We choose $\frac{1}{10}$ and $\frac{9}{10}$, again to make the argument work for $c < \frac{4}{7}$.)

In case 1, since $K_{sep}(X)$ is defined so that $\frac{|X \cap V(K_{sep}(X))|}{|V(K_{sep}(X))|} \geq f(n)$,

$$\frac{|X \cap K_{sep}(X)|}{|V(\mathrm{Ds}(n))|} \geq \frac{|V(K_{sep}(X))|}{|V(K_{big})|} \cdot \frac{|V(K_{big})|}{|V(\mathrm{Ds}(n))|} \cdot f(n) \geq \frac{1}{10} \cdot \frac{8}{9} \cdot f(n) = \Omega(f(n)) = \Omega(\frac{1}{n}).$$

In this case we are done.

In case 2, Lemma 30 implies that there exists a connected component A_k in every $\mathrm{Kn}(n,k) \subseteq K_{conn}(X)$ of size at least $(\frac{5}{7} - O(\frac{1}{n}))|V(\mathrm{Kn}(n,k))|$, and that every pair A_k and A_l are mutually connected. This implies that $\mathrm{Ds}(n) \setminus X$ has a connected component A such that

$$\frac{|V(A)|}{|V(\mathrm{Ds}(n))|} \geq (\frac{5}{7} - O(\frac{1}{n}))\frac{|V(K_{conn}(X))|}{|V(\mathrm{Ds}(n))|} \geq (\frac{5}{7} - O(\frac{1}{n}))(\frac{9}{10})(\frac{|V(K_{big})|}{|V(\mathrm{Ds}(n))|}$$

$$\geq (\frac{5}{7} - O(\frac{1}{n}))(\frac{9}{10})(\frac{8}{9}) > \frac{4}{7} > c.$$

This contradicts the assumption that X is a c-separator for $\mathrm{Ds}(n)$. ◀

To show that $\mathrm{tw}(H_4^n) = \Omega(\mathrm{tw}(\mathrm{Ds}(n)))$, we first show that the treewidth of the generalized pegset intersection graph G_4^n defined earlier is at least that of $\mathrm{Ds}(n)$, then that $\mathrm{tw}(H_4^n) = \Omega(\mathrm{tw}(G_4^n))$. Both of these are accomplished via haven mappings (Lemmas 31 and 35) of a similar flavor to Lemma 15.

▶ **Lemma 31.** $\mathrm{tw}(G_4^n) = \Omega(\mathrm{tw}(\mathrm{Ds}(n)))$.

We prove Lemma 31 as a special case of a more general claim, Lemma 33, about the treewidth of the *tensor product* of graphs:

▶ **Definition 32.** *The tensor product of graphs G and H, denoted $G \times H$, is the graph whose vertex set is the Cartesian product $V(G) \times V(H)$, and whose edges are the pairs of (u_1, v_1) and (u_2, v_2) whose first and second components share edges in $E(G)$ and $E(H)$ respectively, i.e.*

$$\{((u_1, v_1), (u_2, v_2)) \mid u_1, u_2 \in V(G), v_1, v_2 \in V(H), (u_1, u_2) \in E(G), (v_1, v_2) \in E(G)\}.$$

We prove the following:

▶ **Lemma 33.** *Let G and H be connected graphs, and suppose that H is not bipartite. Then*

$$\mathrm{tw}(G \times H) \geq \mathrm{tw}(G).$$

To prove Lemma 33, we first define an association between the vertices of G and those of $J = G \times H$.

▶ **Definition 34.** *Given the tensor product $J = G \times H$ of graphs G and H, define $f : V(J) \to V(G)$ so that for all $(u, v) \in V(J)$,*

$$f((u, v)) = u.$$

Define $g : V(G) \to \mathcal{P}(V(J))$ so that for all $u \in V(G)$,

$$g(u) = f^{-1}(u) = \{(u, v) \mid v \in V(H)\}.$$

We use this definition to prove Lemma 33. The proof is similar in spirit to the proof of Lemma 15:

Proof of Lemma 33. For the lower bound $\mathrm{tw}(G \times H) \geq \mathrm{tw}(G)$, by Lemma 3, G has a haven ϕ of order $k = \mathrm{tw}(G) + 1$. We construct a haven ψ in $J = G \times H$ of order $k' \geq k$, from which the lemma follows. To define ψ, we extend the domains of f and g to sets of vertices in the natural way. That is, for every $X \subseteq V(J)$, let $f(X)$ be the image of all vertices in X under f. For every $Y \in V(G)$, let $g(Y)$ be the union of the images under g of all vertices in Y.

For all $X \subseteq V(J)$, let $\psi(X)$ be the connected component in $J \setminus X$ containing $g(\phi(f(X)))$. (Let $\psi(\emptyset) = \emptyset$.) It suffices to show that:

1. $Y' = g(Y)$ is a nonempty connected component in $J \setminus X$ whenever Y is a nonempty connected component in $G \setminus f(X)$,
2. for all $X \subseteq Z \subseteq V(J)$, $\psi(Z) \subseteq \psi(X)$, and
3. for all $X \subseteq V(J)$, $|f(X)| \leq |X|$.

For (1), suppose Y is a connected component in $G \setminus f(X)$ for some $X \subseteq V(J)$. Let $Y' = g(Y)$. If $|Y| > 1$, then consider any edge $(u, w) \in Y$. Then for every pair of vertices $v, x \in V(H)$, the vertices (u, v) and (w, x) are connected by a path P' in Y'. To construct this path, consider any walk P along a sequence of vertices $(v, z_1, z_2, \ldots, z_l, x)$ of odd length in H from v to x. Such a walk must exist since H is not bipartite, i.e. contains an odd cycle. Construct the corresponding path P' in Y' by alternating between copies of u and copies of w. That is, let

$$P' = ((u, v), (w, z_1), (u, z_2), (w, z_3), \ldots, (w, z_{l-1}), (u, z_l), (w, x)).$$

Since such a path exists for every edge $(u, v) \in Y$, and Y is connected, Y' is also connected.

We deal with the degenerate case $|Y| = 1$ by letting Y' be a single copy (u, v) of the vertex $u \in Y$, and obtain ψ by extending this copy to a connected component.

For (2), it follows from the definition of f and the fact that ϕ is a haven, that $\phi(f(Z)) \subseteq \phi(f(X))$. ψ merely extends $\phi(f(Z))$ and $\phi(f(X))$ to connected components in $J \setminus Z$ and $J \setminus X$ respectively. The connected component B in $J \setminus X$ containing $\phi(f(Z))$ is the same as the connected component in $J \setminus X$ containing $\phi(f(X))$, since both $\phi(f(Z))$ and $\phi(f(X))$ are connected and one is a subset of the other. Furthermore, since $X \subseteq Z$, $J \setminus Z \subseteq J \setminus X$, so removing the additional vertices in $Z \setminus X$ from B cannot result in a connected component with vertices missing from B. That is, the connected component $\psi(Z)$ in $J \setminus Z$ containing $\phi(f(Z))$ is a subset of the connected component $\psi(X)$ in $J \setminus X$ containing $\phi(f(X))$.

(3) is immediate from the definition of f. ◀

Lemma 31 immediately follows from Lemma 33 and the fact that G_4^n is isomorphic to $\mathrm{Ds}(n) \times K_4$.

▶ **Lemma 35.** $\mathrm{tw}(H_4^n) = \Omega(\mathrm{tw}(G_4^n))$.

Proof. Construct a haven mapping analogous to the mapping in Lemma 15. In Lemma 15 we defined f and g as, respectively, mapping sets of configurations to the regular pegsets to which they belong, and mapping sets of regular pegsets to the unions of their configurations. Extend the codomain of f and the domain of g, beyond regular pegsets, to the set of all pegsets in G_4^n. The rest of the argument is similar to the proof of Lemma 15. Again we need to check the following conditions:

1. for all $X \subseteq V(H_p^n)$, $\psi(X)$ is well-defined—i.e. $g(\phi(f(X)))$ is connected and nonempty whenever $\phi(f(X))$ is nonempty,
2. for $Z \subseteq V(H_p^n)$, $\psi(Z) \subseteq \psi(X)$ whenever $X \subseteq Z$, and
3. $|X| = \Omega(f(X))$.

(3) is easy since every configuration belongs to at most four pegsets. The reasoning for (1) is identical to that in the proof of Lemma 15. For (2), the reasoning is also the same. ◀

Theorem 17 follows from Lemma 21, Lemma 31, and Lemma 35.

An Open Pouring Problem

Fabian Frei
Department of Computer Science, ETH Zürich, Switzerland
fabian.frei@inf.ethz.ch

Peter Rossmanith
Department of Computer Science, RWTH Aachen, Germany
rossmani@cs.rwth-aachen.de

David Wehner
Department of Computer Science, ETH Zürich, Switzerland
david.wehner@inf.ethz.ch

Abstract

We analyze a little riddle that has challenged mathematicians for half a century.

Imagine three clubs catering to people with some niche interest. Everyone willing to join a club has done so and nobody new will pick up this eccentric hobby for the foreseeable future, thus the mutually exclusive clubs compete for a common constituency. Members are highly invested in their chosen club; only a targeted campaign plus prolonged personal persuasion can convince them to consider switching. Even then, they will never be enticed into a bigger group as they naturally pride themselves in avoiding the mainstream. Therefore each club occasionally starts a campaign against a larger competitor and sends its own members out on a recommendation program. Each will win one person over; the small club can thus effectively double its own numbers at the larger one's expense.

Is there always a risk for one club to wind up with zero members, forcing it out of business? If so, how many campaign cycles will this take?

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains; Theory of computation → Representations of games and their complexity

Keywords and phrases Pitcher Pouring Problem, Water Jug Riddle, Water Bucket Problem, Vessel Puzzle, Complexity, Die Hard

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.14

Acknowledgements We would like to thank the anonymous reviewers who carefully read this paper for their detailed feedback.

1 The Same Old Pouring Problem Again[1]

Almost anyone who is even remotely fond of logical puzzles and many others have heard of and even solved the following problem:

> *Given two pitchers of three and five ounces capacity and an infinite water supply, can you precisely measure four ounces?*

This already popular pitcher-pouring problem has gained increased prominence when it was featured in the third installment of the Die Hard movie series, released in 1995. The two protagonists John and Zeus are forced to figure out the solution to the problem within five minutes to defuse a bomb. Frantically discussing the problem in detail, they eventually succeed and prevent the explosion with mere seconds left, as expected.

[1] Please patiently pardon particularly peculiar pour puns.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 14; pp. 14:1–14:9
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An outpouring of papers discussing different aspects of the problem ensued – considering the natural generalization to arbitrary capacities of the two pitchers, of course. It also has become somewhat of a pet problem in Artificial Intelligence [8, 6]. The focus was often on didactical aspects of the problem as the solution is rather simple from a mathematical standpoint: The number of liters that can be measured using two given pitchers are exactly the greatest common divisor of the two capacities and its multiples. The complexity of the problem – that is, the number of steps required to solve all instances of the problem with a given total capacity – was first considered and immediately shown to be linear directly following the movie release in 1995 as well [5].

Fortunately, there is a far more challenging pouring problem that is still open.

2 Our Problem: Significantly Less Pouring

The fifth problem of the fifth All-Union round of the Soviet Mathematics Olympiad, held in Riga in 1971, reads as follows:

> В три сосуда налито по целому числу литров воды. В любой сосуд разрешается перелить столько воды, сколько в нем уже содержится, из любого другого сосуда. Докажите, что несколькими такими переливаниями можно освободить один из сосудов. (Сосуды достаточно велики: каждый может вместить всю имеющуюся воду.) [7]
>
> *An integer number of liters of water has been poured into each of three vessels. It is allowed to pour into each vessel as much water as it already contains from an arbitrary other vessel. Prove that several such pourings can empty one of the vessels. (The vessels are sufficiently large: Each one can contain the entire available water.)*

Two small clarifications might be in order. First, each of the three vessels may contain a different natural number of liters of course; the puzzle would be trivial otherwise. Second, it is only implicit that we can never pour water out of a vessel that contains less water than the receiving one. See Figure 1 for an illustrating example with a simple instance and its optimal solution.

The exam designer clearly made an effort to keep the task from being too abstract by casting it into this vessel form. Nevertheless, coming up with a way to perform such magical pouring steps that allow us to double a vessel's content – without additional materials that would render the entire enterprise obsolete – seems to be the hardest challenge here by far.

However, even when freeing ourselves from the pour setting, it remains a tough task to find any natural situation where the described situation might arise. While we gave it our best try in the abstract, the issue is usually skirted altogether, as we will see.

The puzzle made an honoring appearance as the final task in the 54th iteration of the William Lowell Putnam Mathematical Competition. The organizers of the most prestigious mathematical competition worldwide opted for a purely abstract description:

> *Let S be a set of three, not necessarily distinct, positive integers. Show that one can transform S into a set containing 0 by a finite number of applications of the following rule: Select two of the three integers, say x and y, where $x \leq y$ and replace them with $2x$ and $y - x$.* [1]

Two decades later, when the time-proven problem was presented as a challenge to IBM researchers, their puzzlemaster embedded it in a contrived betting game [2]. Most recently, the task took yet another, now overtly magical guise in Germany's 38th National Computer Science Competition [3]. This time around, the participants were not asked to solve the problem mathematically, but had to write a program solving it optimally instead.

Now, this problem was not only foisted upon defenseless exam takers; it has also been included into a carefully curated anthology of mathematical riddles aimed at every avid enigma aficionado [10, 9, 11]. In "Mathematical Puzzles: A Connoisseur's Collection," the puzzle-pondering professor Peter Winkler, a well-respected mathematician and computer scientist, presents the problem in its original form with three water buckets. He goes on to describe his solution to the problem, which might well have been the intended one. His approach guarantees that an initial configuration with n liters in total can be turned into one with an empty bucket in at most $\mathcal{O}(n^2)$ steps.

This result was independently improved upon by two persons with whom Peter Winkler had been sharing the puzzle, Svante Janson from Uppsala University and Garth Payne from Pennsylvania State University. They both described an algorithm that can empty one of the buckets in at most $\mathcal{O}(n \log n)$ steps. Winkler concludes: *"As far as I know, no one knows even approximately how many steps are required for this problem."* [9]

The German translation of the book, published in 2008, adds an optimistic conjecture by Michael H. Albert that the minimal number of steps is far lower, namely $\Theta((\log n)^2)$.

3 Our Contribution

In this section, we first improve in Subsection 3.2 the known upper bound from the linearithmic $\mathcal{O}(n \cdot \log n)$ down to $\mathcal{O}((\log n)^2)$, matching Albert's conjecture. In Subsection 3.2, we then go on to give experimental evidence that, on the one hand, exhibits the very peculiar and mathematically interesting behavior of this problem and, on the other hand, strongly suggests that even Albert's conjecture is too pessimistic still: The required number of steps is far more likely to be $\Theta(\log n)$, which we posit as our improved conjecture. In Subsection 3.3, we prove a lower bound that not only matches our conjecture asymptotically but in fact perfectly fits the experimental data for infinitely many n that we analyze more closely in Subsection 3.4. Note that our results leave open a good gap of a single logarithmic factor, lest Winkler's puzzling problem be spoiled entirely for the reader.

We briefly restate the problem in the notation that will be used in our proofs.

> Let $(a, b, c) \in \mathbb{N}^3$ be a triple of nonnegative integers. We are allowed to perform the following modification step on any triple: Pick from it any two numbers x and y such that $x \leq y$ and then replace them by $2x$ and $y - x$, respectively.
> For any $n \in \mathbb{N}$, denote by $N(n)$ the minimum number of such steps that allows us to transform any given triple $(a, b, c) \in \mathbb{N}^3$ satisfying $a + b + c = n$ into a triple containing a zero. Prove good upper and lower bounds on $N(n)$.

3.1 Upper Bound

We directly state and then prove our upper bound.

▶ **Theorem 1.** *The optimal number of steps required to solve any instance with a total liter count of n is bounded by $N(n) \leq (\log n)^2$.*[2]

Proof. We describe an algorithm that transforms any given configuration $(a, b, c) \in \mathbb{N}^3$ into one containing a zero in at most $(\log n)^2$ steps, where $n = a + b + c$. Our algorithm works in rounds. We may assume without loss of generality that every round starts with

[2] Throughout this paper, log denotes the logarithm to base 2.

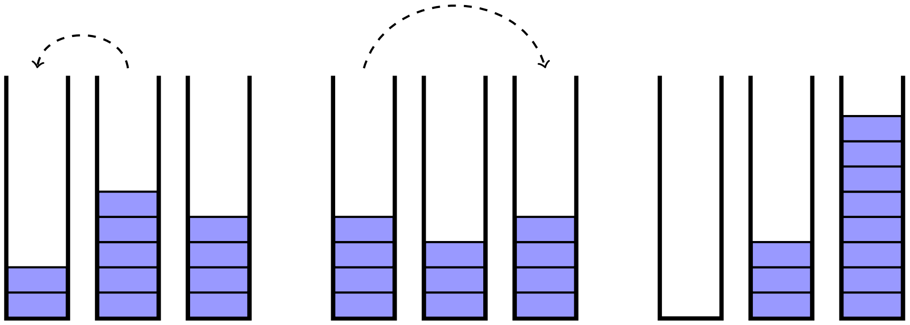

(a) Initial Configuration. **(b)** One pouring later. **(c)** Problem solved.

Figure 1 Optimal solution for the initial instance $(2, 5, 4)$. The total number of liters is $n = 11$, the required number of pouring steps is $N(n) = 2$.

a configuration (a, b, c) that is ordered such that $0 < a \le b \le c$. In every round, it will transform the configuration (a, b, c) into a new configuration (a', b', c') satisfying $a' \le a/2$, using a series of at most $\log n + 1$ steps. As any configuration in ascending order, the initial configuration satisfies $a \le n/3$. It is thus guaranteed that after at most $\log n - 1$ rounds we reach a configuration whose smallest number is at most $n/3 \cdot 2^{1-\log n} = 2/3$, which just means it is zero, as required. We note that Svante's algorithm as described by Winkler [9] is structured in rounds as well. Instead of halving the number a in each round, it only guarantees a decrease by at least 1, however. This crucial difference allows us to improve the upper bound from $\mathcal{O}(n \cdot \log n)$ to $\mathcal{O}((\log n)^2)$.

We now describe a single round that starts with a configuration (a, b, c) satisfying $0 < a \le b \le c$. Let $r := b/a$ be the ratio between the two smallest numbers. We round this ratio both ways and denote the resulting integers by $p := \lfloor r \rfloor$ and $q := \lceil r \rceil$. Let $p_k \dots p_0$ and $q_\ell \dots q_0$ be the minimal binary representations of p and q, respectively; that is, we have $p_0, \dots, p_{k-1}, q_0, \dots q_{\ell-1} \in \{0, 1\}$ and $p_k = q_\ell = 1$ for $k = \lfloor \log p \rfloor$ and $\ell = \lfloor \log q \rfloor$ with $\sum_{i=0}^{k} p_i 2^i = p$ and $\sum_{i=0}^{\ell} q_i 2^i = q$. Note that $0 \le b - pa < a$ and $0 \le qa - b < a$. This implies that $\min\{b - pa, qa - b\} \le a/2$ since $(b - pa) + (qa - b) = (q - p)a \le a$. We can thus consider the following two, potentially overlapping cases.

Case 1: Assume first that $b - pa \le a/2$. In this case we perform $k + 1$ steps for $i = 0, \dots, k$ that will always double what was initially the smallest number a in the triple. To do this, the algorithm has to subtract first a, then $2a$, and generally $2^i a$ from one of the other two numbers in the triple. We use p to decide which one: If $p_i = 1$, we subtract $2^i a$ from the second number, which initially is b; otherwise we have $p_i = 0$ and subtract from the third number, which is c initially. After performing these $k + 1$ steps, the second number and third number will be $b - \sum_{i=0}^{k} p_i 2^i a = b - pa \le a/2$ and $c - \sum_{i=0}^{k} (1 - p_i) 2^i a$, respectively. We have to prove that both the b and c of the initial configuration are sufficiently large so as to never become negative and thus make all steps in this round valid. For b, we can simply use our general observation $b - pa \ge 0$. For c, we have

$$c - \sum_{i=0}^{k} (1 - p_i) 2^i a = c - \sum_{i=0}^{k-1} (1 - p_i) 2^i a = c - \left(\sum_{i=0}^{k-1} 2^i - \sum_{i=0}^{k-1} p_i 2^i \right) a \ge c - 2^k a \ge 0,$$

where the last inequality follows from $k = \lfloor \log p \rfloor$ and $a \le c$. We conclude that this round is feasible and results in a triple whose smallest number is at most $b - pa \le a/2$, as required. The number of steps in this round is exactly $k + 1 = \lfloor \log p \rfloor + 1 \le \lfloor \log q \rfloor + 1$.

Case 2: We now assume $qa - b < a/2$. In this case, we first perform the following ℓ steps for $i = 0, \ldots, \ell - 1$: We always double the first number a in the triple, as we did in the first case, but now we will be subtracting the necessary amount from the second number, the initial b, if $q_i = 1$ and from the third number, the initial c, if $q_i = 0$. After these ℓ steps, the first number of the resulting triple will be $2^\ell a$, the second one $b - \sum_{i=0}^{\ell-1} q_i 2^i a = b - (q - 2^\ell)a$ and the third one $c - \sum_{i=0}^{\ell-1}(1 - q_i)2^i a$. Again, we have to prove that these ℓ steps are in fact possible by showing that b and c are large enough. This is the case because, on the one hand, we have $q \leq p + 1$ and thus $b - (q - 2^\ell)a \geq b - (q - 2^0)a \geq b - pa \geq 0$ and, on the other hand, we have

$$c - \sum_{i=0}^{\ell-1}(1 - q_i)2^i a \geq c - \sum_{i=0}^{\ell-1} 2^i a = b - (2^\ell - 1)a \geq b - (q - 1)a \geq b - pa \geq 0.$$

We now perform the last, $(\ell + 1)$st step of this round. It doubles the second number $b - (q - 2^\ell)a = b - qa + 2^\ell a$ and subtracts the corresponding amount from the first number, which currently is $2^\ell a$. This step is valid since $2^\ell a - (b - qa + 2^\ell a) = qa - b \geq 0$. The round ends with this step and the first number of the triple is now $qa - b < a/2$. The number of steps in this round is precisely $\ell + 1 = \lfloor \log q \rfloor + 1$.

We have shown for both cases how to perform a valid round of at most $\lfloor \log q \rfloor + 1 \leq \lfloor \log(n/1) \rfloor + 1 \leq 1 + \log n$ steps that result in a triple whose smallest number is at most $\lfloor a/2 \rfloor$. As already mentioned, it now suffices to iterate this entire process for at most $(\log n) - 1$ rounds and we end up with a final configuration whose smallest number is 0. The total number of steps over all rounds is thus at most $(\log(n) - 1) \cdot (\log(n) + 1) \leq (\log n)^2$, which proves the theorem. ◀

3.2 Experimental Evidence

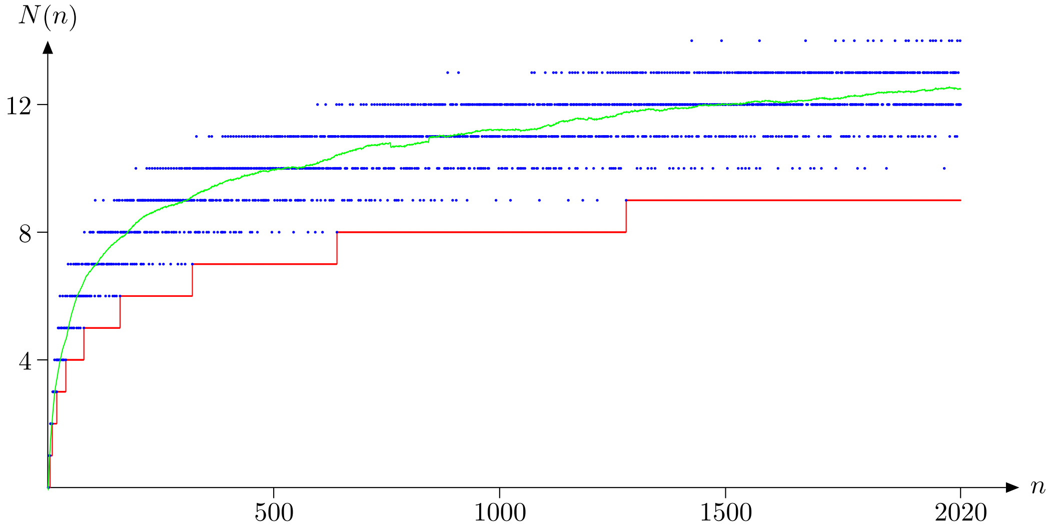


Figure 2 For any blue point, the ordinate indicates the minimum number of pouring steps required in the worst case for a starting configuration (a, b, c) with $n = a + b + c$, where n is given by the abscissa. The green line shows for each n the average of the 85 blue points $(n - 42, \ldots, n + 42)$. The red line plots our lower bound derived in Subsection 3.3.

Table 1 All smallest hard instances for the listed minimum number of steps, that is, all worst-case instances for the smallest n yielding a new step number $N(n)$ for n up to 2020. The values of n are also found as sequence A256001 in the online encyclopedia of integer sequences [4].

$N(n)$	$n = a + b + c$	(a, b, c)
1	3	$(1, 1, 1)$
2	6	$(1, 2, 3)$
3	11	$(1, 4, 6)$
4	15	$(4, 5, 6)$, $(3, 4, 8)$, $(2, 5, 8)$
5	23	$(3, 8, 12)$
6	27	$(5, 9, 13)$
7	45	$(4, 15, 26)$
8	81	$(8, 27, 46)$
9	105	$(27, 35, 43)$, $(8, 35, 62)$, $(8, 27, 70)$
10	195	$(57, 65, 73)$, $(8, 78, 109)$, $(4, 78, 113)$, $(8, 73, 114)$, $(8, 65, 122)$, $(4, 66, 125)$, $(8, 57, 130)$, $(4, 33, 158)$
11	329	$(4, 130, 195)$
12	597	$(175, 199, 223)$
13	885	$(101, 295, 489)$
14	1425	$(206, 475, 744)$

In order to develop a proper intuition of the behavior of the step complexity $N(n)$, we wrote a program that calculates $N(n)$ for any given n. It does so by exhaustive enumeration of all instances and then trying out all feasible solutions. We show our findings in Figure 2. The blue points plot $N(n)$, the step number required for a worst-case triple whose numbers sum up to n, against this total for n from 0 up to 2020.

Across the entire spectrum, we observe erratic jumps up and down. The values where these jumps occur do not seem to follow any discernible pattern, however. Despite their best efforts, the authors were indeed unable to detect any stable structure, except for a small detail that we will discuss later on.

From a global perspective, taking a step back and squinting a little bit, a clear consistent picture emerges out of the confusing micro-behavior: The bulk of the values clearly follows a logarithmic line; we can approximate it by the green line, which plots the average value of $N(n)$ across the interval $[n - 42, n + 42]$. The oscillations of the blue points around the imagined center of gravity begin very small, but grow in amplitude with increasing n. The plotted pairs (n, N) where the amplitude first increases over the previous threshold are the following: From $(15, 4)$ to $(16, 3)$, from $(26, 4)$ to $(27, 6)$, from $(105, 9)$ to $(112, 6)$, and from $(885, 12)$ to $(896, 8)$. The gaps from one threshold to the next appear to be growing exponentially, but again the data is too noisy to deduce any rule.

Another point of interest might be the first values for n at which $N(n)$ spikes up to a new height for the first time. In order to describe the optimal monotone upper bound for this problem, we would need to understand these values. However, they do not exhibit any clear pattern either, besides an approximately exponential growth. The values of n up to 2020 for which $N(n)$ reaches a previously unattained value and the corresponding worst-case instances are displayed in Table 1.

Once more, neither an underlying nor an overarching pattern was to be found, neither in the instances themselves nor in the optimal solutions' step sequences.

To address the complementary question about the optimal monotone *lower* bound, we should at least know what the largest values n keeping $N(n)$ at any fixed value are. Seeing how we have been pouring out all the intractabilities of our problem on the reader, it might be surprising that we can in fact give a quite concise answer to this last question: For any $\ell \in \mathbb{N} \setminus \{0\}$, the largest n to yield $N(n) = \ell$ is $n = 5 \cdot 2^\ell$ and the instance

$$\left(\left\lfloor \frac{5}{3} \cdot 2^\ell \right\rceil - 1, \left\lfloor \frac{5}{3} \cdot 2^\ell \right\rceil, \left\lceil \frac{5}{3} \cdot 2^\ell \right\rceil + 1 \right),$$

where by $\lfloor x \rceil$ we denote x rounded to the nearest integer, emerged as the corresponding unique worst-case instance. We will investigate these instances closer in Subsection 3.4.

3.3 Lower Bound

We finally prove our lower bound and show how well it matches our experimental data.

▶ **Theorem 2.** *The number of steps for solving a worst-case instance with a total liter count of n is at least $\lceil \log((n+1)/5) \rceil = \Omega(\log n)$.*

Proof. Let $t := \frac{n}{3}$ and consider the following configuration (a, b, c) depending on the remainder of n modulo 3:

$$(a,b,c) = \begin{cases} (t-1, t, t+1), & \text{if } t - \lfloor t \rfloor = 0\,,\ \text{i. e., } n \equiv 0 \pmod 3 \\ (t - \frac{4}{3}, t - \frac{1}{3}, t + \frac{5}{3}), & \text{if } t - \lfloor t \rfloor = \frac{1}{3},\ \text{i. e., } n \equiv 1 \pmod 3 \\ (t - \frac{5}{3}, t + \frac{1}{3}, t + \frac{4}{3}), & \text{if } t - \lfloor t \rfloor = \frac{2}{3},\ \text{i. e., } n \equiv 2 \pmod 3. \end{cases}$$

We can write this down in general as $(t+d_1, t+d_2, t+d_3)$ with $d_1 < d_2 < d_3$. After one step, we have either $(2t+2d_1, d_2-d_1, t+d_3)$ or $(2t+2d_1, t+d_2, d_3-d_1)$ or $(t+d_1, 2t+2d_2, d_3-d_2)$. By re-ordering, all of these configurations can be written as $(x_1, t+y_1, 2t+z_1)$. We will generally write the configuration after i steps as $(x_i, t+y_i, 2t+z_i)$. Let u_i, v_i, and w_i be the absolute values of x_i, y_i, and z_i in ascending order, that is, we always have $\{u_i, v_i, w_i\} = \{|x_i|, |y_i|, |z_i|\}$ and $u_i \le v_i \le w_i$. For $i = 1$, we can directly verify that $w_i \le 5/3 \cdot 2$ and $v_i \le 5/3 \cdot 2 - 1/3$. In general, we observe that the largest absolute value after step i, namely w_i, can be at most double of what the previously largest absolute value was; we have $w_i \le 2w_{i-1}$. Moreover, the second largest absolute value v_i can be at most the sum of the two largest absolute values in the preceding step; we have $v_i \le v_{i-1} + w_{i-1}$. By induction, we therefore obtain $u_i \le v_i \le w_i \le 5/3 \cdot 2^i$ and $v_i \le 5/3 \cdot 2^i - 1/3$ for the absolute values after the ith step. It immediately follows that

$$v_i + w_i \le 5/3 \cdot 2^{i+1} - 1/3. \tag{1}$$

Clearly, as long as $v_i + w_i < t$, no two of the three numbers can add up to t and thus a, b, c cannot be equal. However, the only way to reach a configuration that contains a zero is from a configuration with two equal numbers. Thus, $v_i + w_i \ge t = n/3$ is a necessary condition for $(x_i, t+y_i, 2t+z_i)$, the configuration after step i, to contain two equal numbers. Using the bound (1) derived above, we conclude that this condition will not be met as long as the following two equivalent inequalities remain true:

$$\frac{5}{3} \cdot 2^{i+1} - \frac{1}{3} < \frac{n}{3} \qquad \Longleftrightarrow \qquad i + 1 < \log \frac{n+1}{5}.$$

Consequently, any step number k that affords us just a chance of ending up with two equal numbers has to satisfy $k + 1 \ge \log((n+1)/5)$. The number of steps k being an integer, we can improve this to $k \ge \lceil \log((n+1)/5) \rceil - 1$. Only after at least k steps we might have two equal number appearing in our configuration. One additional step involving these two numbers is then required to produce a zero. Therefore, the minimum number of steps is at least $\lceil \log((n+1)/5) \rceil = \Omega(\log n)$. ◀

We would now like to present evidence that this bound is in fact optimal for infinitely many values of n in the following section.

3.4 Solving Hard Instances Optimally

We complement the lower bound derived in Subsection 3.3 with an analysis of all instances of the form

$$\left(\left\lfloor \frac{5}{3}\cdot 2^\ell \right\rfloor - 1, \left\lfloor \frac{5}{3}\cdot 2^\ell \right\rceil, \left\lceil \frac{5}{3}\cdot 2^\ell \right\rceil + 1\right),$$

where $n = 5 \cdot 2^\ell$ for any natural number ℓ.

For these instances, our lower bound evaluates to $\lceil \log(2^\ell + 1/5) \rceil = \ell + 1$. (Note that these instances have the form $(t-4/3, t-1/3, t+5/3)$ and $(t-5/3, t+1/3, t+4/3)$ for even and odd ℓ, respectively.) We will now show that $\ell+1$ steps are indeed sufficient for solving these instances.

For $\ell = 0$ and $\ell = 1$, $N(n) \geq \ell + 1$ is easily verified by checking all possibilities. Now let $\ell \geq 2$ and $n = 5 \cdot 2^\ell$.

Case 1: Assume that ℓ is odd. Let $k = (\ell-3)/2$. Using binary representation, we can now represent the numbers in our hard instance (a, b, c) as

$$a = 11(01)^k 00_2,$$
$$b = 11(01)^k 01_2, \text{ and}$$
$$c = 11(01)^k 11_2.$$

It is easy to check that these numbers sum up to $n = 101(0)^\ell_2$. We start by transferring from the last number to the middle one, yielding $(a, 2b, c-b) = (a, 2b, 2) = (11(01)^k 00_2, 11(01)^k 010, 10_2)$. We then alternately subtract the last number from the second and the first, beginning with the second, for $2k+2$ steps in total. Finally, we subtract the last number from the first one again. The first number will then be zero after a total of $1 + (2k+2) + 1 = 2k + 4 = \ell + 1$ steps.

Case 2: Assume that ℓ is even. Let $k = (\ell-4)/2$. Using binary representation again, we have

$$a = 11(01)^k 001_2,$$
$$b = 11(01)^k 011_2, \text{ and}$$
$$c = 11(01)^k 100_2.$$

Now, we first go from (a, b, c) to $(2a, b{-}a, c) = (2a, 2, c) = (11(01)^k 0010_2), 10_2, 11(01)^k 100_2)$. We then subtract the middle number from the first and then from the last. Now we begin to subtract the middle number alternately from the last and the first, beginning with the last, for $2k+1$ steps in total. Finally, we subtract once more the middle number from the last number. This will result in the last number becoming zero after a total of $1 + 2 + (2k+1) + 1 = 2k + 5 = \ell + 1$ steps.

We have thus shown $\log(2n/5)$ to be the optimal bound for infinitely many hard instances.

4 L'Art Pour l'Art

We hope to have sparked in the reader an unquenchable enthusiasm for the presented pouring-pot problem, prompting a perplexing pot-pourri of pertinent papers and perceptive proofs from our prodigious puzzle partners.

References

1 URL: `https://kskedlaya.org/putnam-archive/1993.pdf`.

2 URL: `https://www.research.ibm.com/haifa/ponderthis/challenges/May2015.html`.

3 URL: `https://bwinf.de/fileadmin/bundeswettbewerb/38/BwInf38-Aufgabenblatt.pdf`.

4 URL: `https://oeis.org/A256001`.

5 Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Measuring with jugs. *Theor. Comput. Sci.*, 282(2):259–270, 2002.

6 Yiu-Kwong Man. A non-heuristic approach to the general two water jugs problem. *Theor. Comput. Sci.*, (10):904–908, 2013.

7 Николай Борисович Васильев (Nikolaj Borisovich Vasil'ev) and Александр Александрович Егоров (Aleksandr Aleksandrovich Egorov). *Задачи всесоюзных математических олимпиад (Zadachi Vsesojuznyh Matematicheskih Olimpiad, Problems of the All-Union Mathematical Olympiads)*. Наука (Nauka), 1988.

8 Glânffrwd P. Thomas. The water jugs problem: Solutions from artificial intelligence and mathematical viewpoints. *Mathematics in School*, 24(5):34–37, 1995.

9 Peter Winkler. *Mathematical Puzzles: A Connoisseur's Collection*. A K Peters, 2004.

10 Peter Winkler. Five algorithmic puzzles. In *Tribute to a Mathemagician*, pages 109–118. A K Peters, 2005.

11 Peter Winkler. *Mathematische Rätsel für Liebhaber*. Springer, 2008.

Multi-Robot Motion Planning of k-Colored Discs Is PSPACE-Hard

Thomas Brocken
TU Eindhoven, The Netherlands
t.brocken@student.tue.nl

G. Wessel van der Heijden
TU Eindhoven, The Netherlands
g.w.v.d.heijden@student.tue.nl

Irina Kostitsyna
TU Eindhoven, The Netherlands
i.kostitsyna@tue.nl

Lloyd E. Lo-Wong
TU Eindhoven, The Netherlands
l.e.lo-wong@student.tue.nl

Remco J. A. Surtel
TU Eindhoven, The Netherlands
r.j.a.surtel@student.tue.nl

Abstract

In the problem of *multi-robot motion planning*, a group of robots, placed in a polygonal domain with obstacles, must be moved from their starting positions to a set of target positions. We consider the specific case of unlabeled disc robots of two different sizes. That is, within one class of robots, where a class is given by the robots' size, any robot can be moved to any of the corresponding target positions. We prove that the decision problem of whether there exists a schedule moving the robots to the target positions is PSPACE-hard.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Disc-robot motion planning, algorithmic complexity, PSPACE-hard

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.15

1 Introduction

Due to a wide range of applications, the multi-robot motion planning problem has received a great amount of attention in the theoretical computer science community in recent years. In the most general setting, the problem can be phrased in the following way: given a set of robots placed in a polygonal domain, find a schedule to move the robots from their initial locations to some specified target locations without collisions. From the point of view of identifying which robots move to which target positions, we can distinguish between *labeled* and *unlabeled* robot motion planning. Labeled motion planning is the most studied and, possibly, is a more natural variant of the problem. In it the robots have unique IDs, and each robot has a specifically assigned target location. In this paper, however, we are more interested in unlabeled robot motion planning, where the robots are indistinguishable from one another, and each robot can move to any of the specified target locations. A classic example of a motivating application for this problem is a swarm of robots operating in a warehouse, where it does not matter which of the robots arrives to pick up an item to be transported. Generalizing the notions of labeled and unlabeled motion planning, Solovey and Halperin [6] introduce the *k-color* robot motion planning problem, where given are k classes of robots and k sets of target positions. Within each class the robots are unlabeled,

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 15; pp. 15:1–15:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

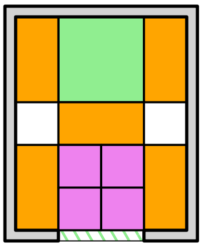


Figure 1 An example of a Sliding Block puzzle. The goal is to take the 2×2 green square outside the box through the exit on the bottom (through which only the green square can slide).

and each robot may move to any location in the corresponding set of target positions. When $k = n$ the problem becomes the standard labeled version of the robot motion planning, and when $k = 1$ it is the unlabeled version.

In this paper we consider the k-color Disc-Robot Motion Planning problem, k-DRMP, where classes of robots differ only by their radii. We show that the problem of deciding whether a particular target location can be reached by a robot from the corresponding class is PSPACE-hard. Our results imply that a version of the Sliding Block game with round pieces can make for a fun and interesting puzzle.

Related work. We start with a brief overview of the known algorithmic results for the disc robot motion planning problem. For unlabeled unit disc robots inside a simple polygon, Adler et al. [1] develop a polynomial-time algorithm to solve the problem under an additional requirement that the distance between any two points from the union of the starting and target locations is at least 4. For unlabeled unit disc robots inside a polygonal domain with obstacles, Solovey et al. [8] show how to find a solution close to optimal in polynomial time. In addition to the same requirement on the separation between the starting/target positions, they require the minimum distance between a robot location and an obstacle to be at least $\sqrt{5}$.

In contrast, for a set of disc robots of possibly different radii in a simple polygon, it is NP-hard to decide whether a target location can be reached by any robot [9].

A wider range of hardness results exists for rectangular or square-shaped robots. Many of these are inspired by *Sliding Block* puzzles, a family of popular games where different shapes are densely packed in a rectangular grid box with little room for movement, and the goal is to free a specific target block and move it outside of the box by sliding the pieces around. Figure 1 shows an example of a puzzle where the blocks are rectangles of integer side length.

One of the earliest results is due to Hopcroft et al. [5], where they show that it is PSPACE-hard to decide whether a given set of rectangular robots enclosed in a rectangular domain can be reconfigured into a particular target configuration. Flake and Baum [3] showed that solving the *Rush Hour* puzzle on an $n \times n$ grid is PSPACE-complete. Rush Hour is a type of a sliding-block puzzle, where the blocks, called cars, are rectangles of width 1 and of length either 2 or 3, and the cars are only allowed to move parallel to their longer side. To prove the hardness of the Rush Hour puzzle, Flake and Baum develop a new specialized model of computation based on “Generalized Rush Hour” logic. They show how to simulate a Finite Turing Machine with circuits built on this logic, which settles the complexity of the problem.

Inspired by Flake and Baum’s construction, Hearn and Demaine [4] develop a *Non-deterministic Constraint Logic* (NCL) framework which has proven to be invaluable in showing hardness results for many problems, based on puzzles and otherwise. To showcase the power of the NCL, they use it to prove PSPACE-completeness of a number of puzzles, including the Sliding Block, even when all blocks are small (in particular, of size 1 × 2), the classic Rush Hour (for cars of length 2 and 3), and others. Tromp and Cilibrasi [10] used the NCL framework to show that Rush Hour is PSPACE-complete even for the cars of length 2 alone. Finally, using NCL, Solovey and Halperin [7] prove that unlabeled multi-robot motion planning for unit square robots moving amidst polygonal obstacles is PSPACE-hard.

Contribution. The contrast between the abundance of hardness results for rectangular and square robots and a few results, negative as well as positive, for disc robots, suggests that the complexity of the problem greatly depends on the shape of the robots, even when the difference between the shapes is seemingly insignificant. Establishing the complexity of multi-robot motion planning of unit disc robots has been an open problem for quite some time. In this paper we show the first PSPACE-hardness result for motion planning of disc robots. In particular, we show that the 2-color multi-robot motion planning problem for disc robots with radii 1/2 or 1 in a polygonal domain is PSPACE-hard by a reduction from the NCL. In contrast, the NP-hardness construction of [9] uses discs of very different sizes with a large ratio between the largest and the smallest disc.

The rest of the paper is structured in the following way. In Section 2 we introduce a formal problem statement and overview the NCL. In Section 3 we show the hardness reduction. We start with describing the gadgets in Section 3.1, and prove their correctness in Section 3.2. Finally, in Section 3.3 we state our main results.

2 Problem statement and preliminaries

In this section we start with a few definitions, and we state the k-color Disc-Robot Motion Planning (k-DRMP) problem more formally.

Let P be a polygonal domain in the plane. By $D(p, r)$ we denote a disc of radius r centered at a point p. A point $p \in P$ is a *valid* position for a disc robot with radius $r > 0$, if $D(p, r)$ is fully contained in P. A set of points $S = \{p_1, \ldots, p_n\} \subset P$ is a *valid configuration* for a set of robots with radius r if (1) p_i is a valid position for a robot of radius r for all $1 \leq i \leq n$, and (2) discs $D(p_i, r)$ and $D(p_j, r)$ do not intersect in their interior, that is, if $|p_i - p_j| \geq 2r$, for all $1 \leq i < j \leq n$.

For k distinct positive radii $\{r_1, r_2, \ldots, r_k\}$ and k positive integers $\{n_1, n_2, \ldots, n_k\}$, denote a k*-configuration* to be a set of k configuration-radius pairs $\mathcal{S} = \{(S_i, r_i) : |S_i| = n_i\}$. A k-configuration is *valid* if each S_i is a valid configuration for disc robots of radius r_i, and for all $i < j$, all $x \in S_i$ and all $y \in S_j$, the discs of respective radii centered at x and y do not intersect in their interior, that is, $|x - y| \geq r_i + r_j$. We will refer to the set of robots with the same radius as a *class* of robots. Thus, each S_i specifies a configuration of a class of robots with radius r_i.

We say that a set of n disc robots with radius r can be *reconfigured* from a valid configuration S into a valid configuration T, if $|S| = |T| = n$, and there exist n paths $\{\pi_1, \pi_2, \ldots, \pi_n\}$, where each path $\pi_i : [0, 1] \to \mathbb{R}^2$ is a continuous curve, such that their starting points form the set S (i.e., $\bigcup_i \pi_i(0) = S$), their final points form the set T, (i.e., $\bigcup_i \pi_i(1) = T$), and at any moment in time $t \in [0, 1]$, the set of points $\{\pi_1(t), \pi_2(t), \ldots, \pi_n(t)\}$ forms a valid configuration for the given value of r.

Analogously, we say that k classes of robots can be *reconfigured* from a valid k-configuration $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ into a valid k-configuration $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$, if $|S_i| = |T_i|$ for all i, and there exist a set of paths which reconfigure each S_i into T_i, such that no two robots overlap at any moment in time.

Drawing inspiration from Hearn and Demaine [4] and Solovey and Halperin [7], we define a few variants of the k-DRMP problem.

Multi-to-multi k-DRMP Given k classes of robots and two valid k-configurations $\mathcal{S}$ and $\mathcal{T}$, decide whether the robots can be reconfigured from $\mathcal{S}$ to $\mathcal{T}$.

Multi-to-single k-DRMP Given k classes of robots, a valid k-configuration $\mathcal{S}$, and a target position $t \in P$, decide whether there exists a valid k-configuration $\mathcal{T}$ with $t \in T_i$ for some $T_i \in \mathcal{T}$, such that the robots can be reconfigured from $\mathcal{S}$ to $\mathcal{T}$.

Multi-to-single-in-class k-DRMP Given k classes of robots with distinct radii $\{r_1, \ldots, r_k\}$, some $1 \leq i \leq k$, a valid k-configuration $\mathcal{S}$, and a target position $t \in P$, decide whether there exists a valid k-configuration $\mathcal{T}$ with $t \in T_i$, where $T_i \in \mathcal{T}$ is the target configuration for the robots with radius r_i, such that the robots can be reconfigured from $\mathcal{S}$ to $\mathcal{T}$.

Intuitively, the *multi-to-multi* problem can be interpreted as follows: Let k-configurations $\mathcal{S}$ and $\mathcal{T}$ represent the start and target positions respectively for the k classes of robots. Can the robots move from $\mathcal{S}$ to $\mathcal{T}$ without any collisions? In this paper we will prove that this, and the other two variants of the k-DRMP problem, are PSPACE-hard. Note, that it is possible to define more variants of the k-DRMP problem by varying which starting or target positions might be fixed, possibly with a fixed matching on them, with specified robot radii, or in any other way along these lines. Many of them can be shown PSPACE-hard with a slight modification to our reduction.

2.1 Nondeterministic constraint logic

We will now briefly introduce the *nondeterministic constraint logic* (NCL). Hearn and Demaine [4] define an NCL machine as a weighted graph $G = (V, E)$, with nonnegative integer weights on the edges, and with integer *minimum in-flow* constraints on the nodes. A *state* of the NCL machine is an assignment of directions onto the edges of G. A state is *valid* if for every node the total weight of incoming edges is at least the value of the minimum in-flow constraint of that node.

Consider a valid state of the NCL machine, and some edge $e = (u, v) \in E$ directed from u to v. We can perform an *edge flip* by reassigning the orientation of e from v to u, as long as the state after the flip remains valid, that is, the in-flow constraint of u is still satisfied. The edge flip operation describes possible transitions between the states of the NCL machine.

Hearn and Demaine [4] show that, even for very restricted versions of the NCL machine, it is PSPACE-complete to decide whether there exists a sequence of valid edge flips which transforms one valid state into another. In particular, the PSPACE-completeness holds for the following four decision problems on an NCL machine which is (1) defined on a simple planar graph $G = (V, E)$, (2) has edge weights either 1 or 2, (3) has the minimum in-flow constraint 2 on all nodes, and (4) has nodes of types *AND* or *protected OR* (which we describe later). The decision problems are:

State-to-state Given two states σ_1 and σ_2, decide whether there exists a valid sequence of edge flips that transforms σ_1 into σ_2.

State-to-edge Given a state σ_1 and an edge $e \in E$, decide whether there exists a state σ_2 such that e has the opposite orientations in σ_1 and σ_2, and there exists a valid sequence of edge flips that transforms σ_1 into σ_2.

Edge-to-edge Given two edges e_1 and e_2 with specific orientations, decide whether there exist two states σ_1 and σ_2 with e_1 and e_2 of prescribed orientation respectively, and there exists a valid sequence of edge flips that transforms σ_1 into σ_2.

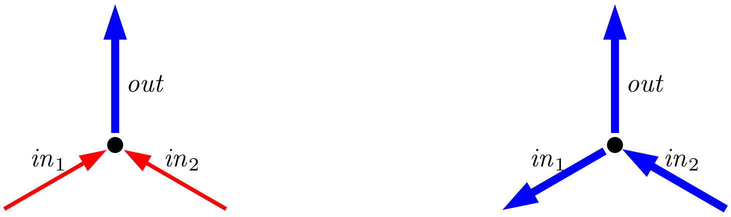


Figure 2 Two types of nodes. Edges with weight 1 are shown in red and edges with weight 2 are shown in blue. Left: An *AND* node. The minimum in-flow requirement of the vertex is 2. Edge *out* can be directed outwards only if both in_1 and in_2 are directed inwards, if the minimum in-flow constraint were to be maintained. Right: A *protected OR* node. Edge *out* can be directed outwards if either in_1 and in_2 is directed inwards. In a *protected OR* it is not possible for both in-edges to be directed inwards simultaneously.

Edge-to-state (symmetric to state-to-edge) Given an edge $e \in E$ and a state σ_2, decide whether there exists a state σ_1 such that e has the opposite orientations in σ_1 and σ_2, and there exists a valid sequence of edge flips that transforms σ_1 into σ_2.

The two types of nodes, the *AND* and the *protected OR*, are both degree three nodes with the following properties (refer to Figure 2). The *AND* node has two incident edges of weight 1, and one incident edge of weight 2. Thus, to satisfy the in-flow constraint, the weight-2 edge can be directed outwards only if both weight-1 edges are directed inwards. All incident edges of an *OR* node have weight 2. Thus, an edge can be directed outwards if at least one other incident edges is directed inwards. In a *protected OR* node two incident edges are labeled as *input*, and one as *output*. The "protected" property forbids the two input edges to be directed inwards at the same time. In many cases, including ours, this restriction simplifies reductions.

▶ **Theorem 1** (Theorem 11 [4]). *State-to-edge, edge-to-state, state-to-state, and edge-to-edge are PSPACE-complete, even when the constraint graph is simple, planar, and only has nodes of types AND and protected OR.*

3 From NCL to 2-DRMP

In this section we will prove that the k-DRMP problem is PSPACE-hard for k classes of unlabeled disc robots moving amidst obstacles constructed out of line segments and circular arcs, even if $k = 2$. We reduce from the NCL problem, for a given constraint graph G we construct a 2-DRMP instance that emulates the NCL machine built on G. Then, by considering the *state-to-edge* and *state-to-state* versions of the NCL problem we will show that the three variants of the 2-DRMP problem defined in Section 2 are PSPACE-hard.

Consider an instance of a constraint graph $G = (V, E)$ with *AND*- and *protected OR*-nodes, and consider its orthogonal drawing on a square grid, with the nodes having integer coordinates, and edges having at most one bend [2] (refer to Figure 3 for an example). We will construct an instance of the 2-DRMP problem such that it is equivalent to the NCL problem on G.

Similarly to the constructions in [3, 4, 7], we create a grid-like environment (refer to Figure 4): square cells of size 18×18 are separated with walls of width 1. The cells correspond to the nodes of G or to the edges passing through grid points. If there is an edge in G passing between two adjacent cells, there is an opening of width 2 in the middle of the wall

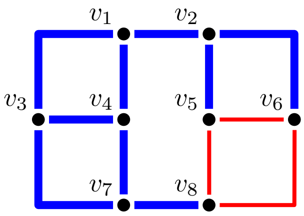


Figure 3 Orthogonal drawing of a constraint graph G. Edges with weight 1 are shown in red, while edges with weight 2 are shown in blue. Nodes v_5, v_6, and v_8 are *AND*-nodes, and nodes v_1, v_2, v_3, v_4 and v_7 are *protected OR*-nodes.

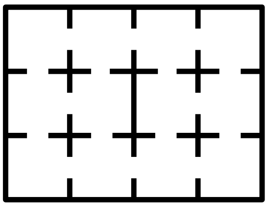


Figure 4 The grid-environment to be filled with gadgets, corresponding to graph G shown in Figure 3. Square cells either correspond to the nodes of G, or grid points that edges pass through.

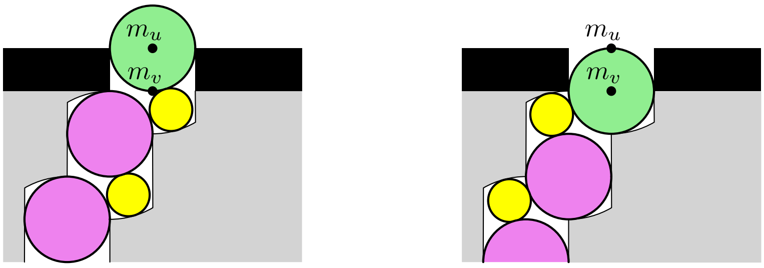


Figure 5 Two terminal positions of an edge robot. Left: edge directed from u to v. Right: edge directed from v to u.

separating the two cells. In each cell we construct free space (by filling its complement with obstacles) and densely place two classes of disc robots with radii 1/2 and 1 to emulate the nodes and the edges of G. The details of the three types of gadgets are described in the following section.

3.1 Gadgets

The three types of gadgets needed to emulate an NCL machine are *AND* gadgets, *protected OR* gadgets, and connector gadgets. Before we describe the specifics of each of the gadgets, we introduce a few building components which will appear in all of them.

Edge robot. Every gadget has two or three length-2 openings in the middle of the wall edges surrounding the square cell of the gadget. Next to each opening a radius-1 *edge* robot is placed (shown in green in the figures), which is shared by the two adjacent gadgets, and which may go back and forth through the opening. The construction of the gadgets is such that only the edge robots can enter and leave their corresponding gadgets. The rest of the robots, called *internal* robots, will always remain within their gadget. Edge robots are restricted in their movement by obstacles either as depicted in Figure 5 or by an obstacle as depicted in Figure 6. Consider an opening between two gadgets corresponding to some edge $(u, v) \in E$. Denote by m_u and m_v the two midpoints on the longer edges of the 1×2 rectangle forming the opening, respectively closer to the gadgets corresponding to u and v (refer to Figure 5). We will call these points *terminal* positions of the edge robot. With respect to a given gadget, we will distinguish between an *inside* terminal position and an *outside* terminal position of an edge robot. The terminal positions of the edge robot will correspond to the orientation of the edge (u, v) in G: the robot centered at m_u corresponds

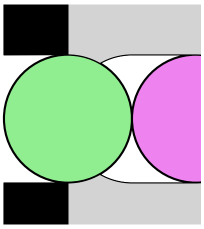


Figure 6 Nudges at an edge robot. The slight narrowing in the free space does not let the interior violet disc to travel more than a unit distance to the left.

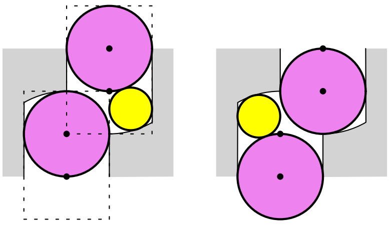


Figure 7 Two configurations of a parallel component.

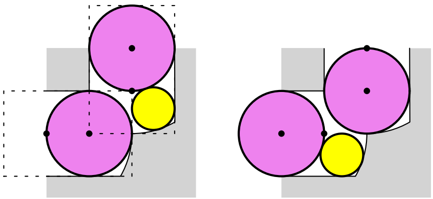


Figure 8 Two configurations of a perpendicular component.

to the edge directed from u to v (Figure 5 (left)), and the robot centered at m_v corresponds to the edge directed from v to u (Figure 5 (right)). Intermediate positions of the robot do not define a specific orientation of the edge, and may correspond to any direction. Thus, the positions of the edge robots, if they are all in terminal positions, will fully describe a state of an NCL machine.

Parallel and perpendicular components. The two constructions shown in Figures 7 and 8 are called a *parallel component* and a *perpendicular component* respectively. The parallel component consists of free space formed by two parallel 2×3 rectangles (outlined with a dashed line) overlapping in a corner 1×1 square. Place two radius-1 discs (violet in the figure) and a radius-(1/2) disc (yellow in the figure) in the resulting free space. For each 2×3 rectangle consider two points placed at a unit distance from three of the four sides of the rectangle. These points are the terminal positions for the two radius-1 discs. Finally, the short unit-length boundary edges of the free space are rounded (replaced with radius-2 arcs centered at the further terminal positions) so that the radius-(1/2) disc touches a radius-1 disc when moving into a corner of a rectangle.

The perpendicular component consists of two perpendicular 2×3 rectangles (in dashed) overlapping in a corner 1×1 square. Similarly, place two radius-1 discs and a radius-(1/2) disc in the free space, and consider the four terminal positions for the radius-1 discs. Again, the short unit length boundary edges are replaced with radius-2 arcs centered at the further terminal positions.

The parallel and perpendicular components are designed for propagating a signal of the position of an edge robot. In Section 3.2 we will show the following property of a chain of parallel and perpendicular configurations. Let $\mathcal{C}$ be a chain of at least 1 parallel and 1 perpendicular configuration, possibly extended with aligned radius-1 discs (for example, as the one used in the connector gadget shown in Figure 11 (right)). Let A^* be the first radius-1 robot in $\mathcal{C}$, and B^* be the last radius-1 robot in $\mathcal{C}$. Let t_1 and t_2 be the terminal positions of A^*, and t_3 and t_4 be the terminal positions of B^*, such that t_1, t_2, t_3, and t_4 appear in order along $\mathcal{C}$. Then the following lemma holds.

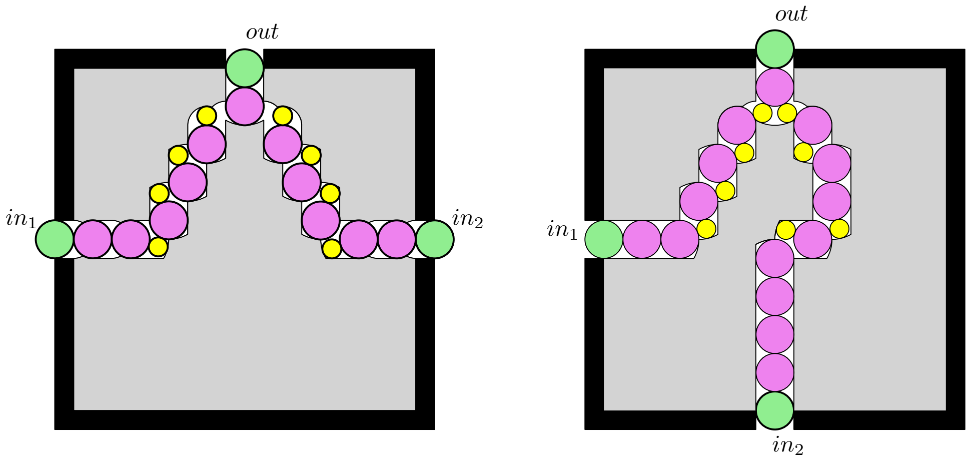


Figure 9 *AND*-gadgets representing *AND* nodes. Edge robot *out* can move inside if and only if both edge robots in_1 and in_2 are outside.

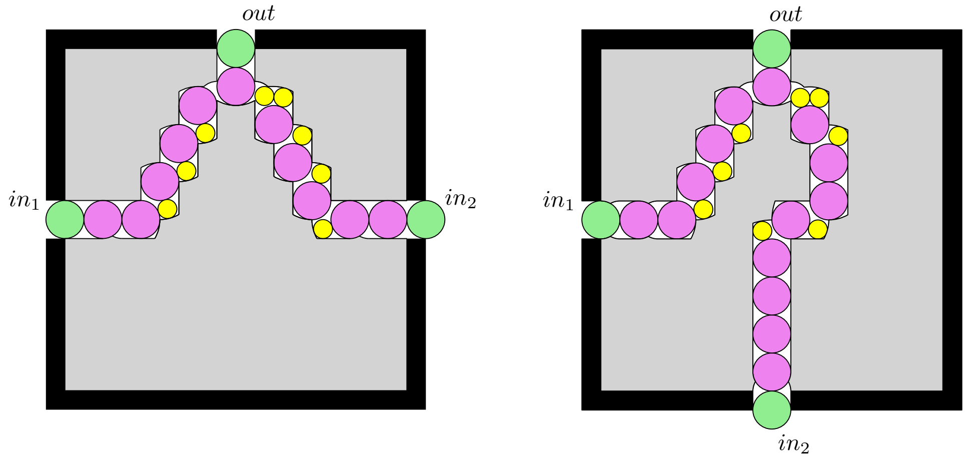


Figure 10 The *protected OR*-gadgets representing *protected OR* vertices. Robot *out* can move inside if and only if either in_1 or in_2 is outside. In this figure, in_2 has moved out allowing robot *out* to move in.

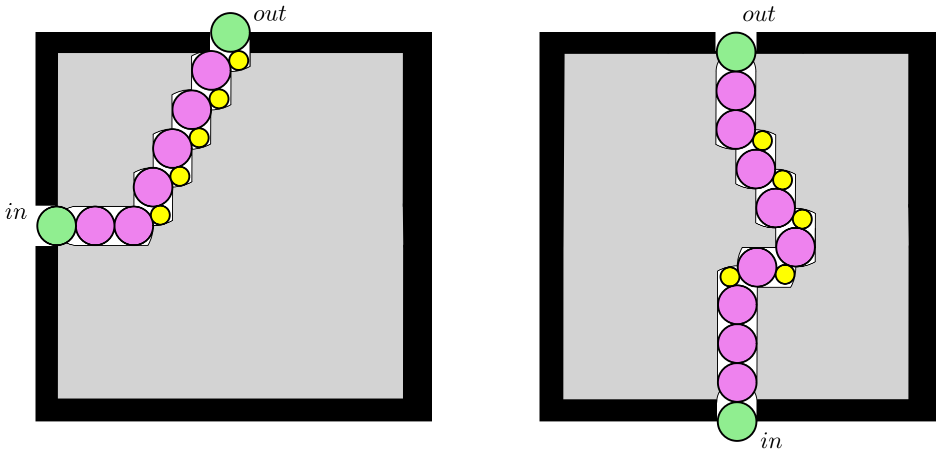


Figure 11 Connector gadgets representing connector vertices. Robot *out* can move inside only if robot *in* is outside and vice versa.

▶ **Lemma 2.** *Robot A^* can move to its terminal position t_2 only if B^* is in its terminal position t_4. Similarly, B^* can move to its terminal position t_3 only if A^* is in its terminal position t_1.*

Using this lemma we will use a sequence of at least 2 parallel and perpendicular components to force the edge robots to be located at one of their terminal positions. We are now ready to show the construction of the gadgets.

AND gadgets. There are two versions of the *AND* gadget (up to mirror symmetry and rotation by 90°, 180°, or 270°), shown in Figure 9. Also refer to Figure 17. They are designed in such a way that the edge robot *out* can only be moved to the inside terminal position if edge robots in_1 and in_2 are both moved to their outside terminal positions. Indeed, the edge robot *out* can move to the inside terminal position only if the two yellow discs of radius 1/2 below it move to the left and to the right, respectively. These yellow discs can move left and right only if both radius-1 violet discs touching them move down, which by Lemma 2 can happen only if both edge robots in_1 and in_2 move to their outside terminal positions. In Section 3.2 we will prove the following lemma.

▶ **Lemma 3.** *In the AND gadgets, the edge robot "out" can be moved to the inside terminal position only if the edge robots "in_1" and "in_2" are both moved to (or beyond) their outside terminal positions. The edge robot "in_1" ("in_2") can be moved to its inside terminal position only if the edge robot "out" is moved to (or beyond) its outside terminal position.*

Protected OR gadgets. Similarly to the *AND* gadgets, there are two versions of the *protected OR* gadget shown in Figure 10. Also refer to Figure 18. The *protected OR* gadgets are designed in such a way that the edge robot *out* can move to the inside terminal position of the gadget if and only if either in_1 or in_2 move to their outside terminal positions. Indeed, the edge disc *out* can move to the inside terminal position only if the yellow discs of radius 1/2 below it move to the left or to the right. These yellow discs can move left or right only if at least one of the adjacent radius-1 violet discs moves one unit down, which by Lemma 2 can happen only if both edge robots in_1 and in_2 move to their outside terminal positions. In Section 3.3 we will prove that the "protected" property of this gadget is preserved. That is, we will show a correspondence between a valid reconfiguration of the robots and a sequence of edge flips in the graph G, such that no two input edges of a *protected OR* in G are both directed inwards at the same time. In Section 3.2 we will prove the following lemma.

▶ **Lemma 4.** *In the protected OR gadgets, the edge robot "out" can be moved to the inside terminal position only if at least one of the edge robots "in_1" and "in_2" is moved to (or beyond) their outside terminal positions. The edge robot "in_1" ("in_2") can be moved to its inside terminal position only if the edge robot "out" is moved to (or beyond) its outside terminal position.*

Connector gadgets. Our last type of gadget, the connector gadget, is shown in Figure 11. The two versions of the connector gadget represent a corner piece of an edge (Figure 11 (left)) and a straight piece of an edge (Figure 11 (right)). The connector gadgets consist of a chain of parallel and/or perpendicular components, possibly extended by a sequence of aligned unit discs. The following lemma follows directly from Lemma 2.

▶ **Lemma 5.** *In the connector gadget, an edge robot can only be moved to its inside terminal position if the other edge robot is moved to (or beyond) its outside terminal position.*

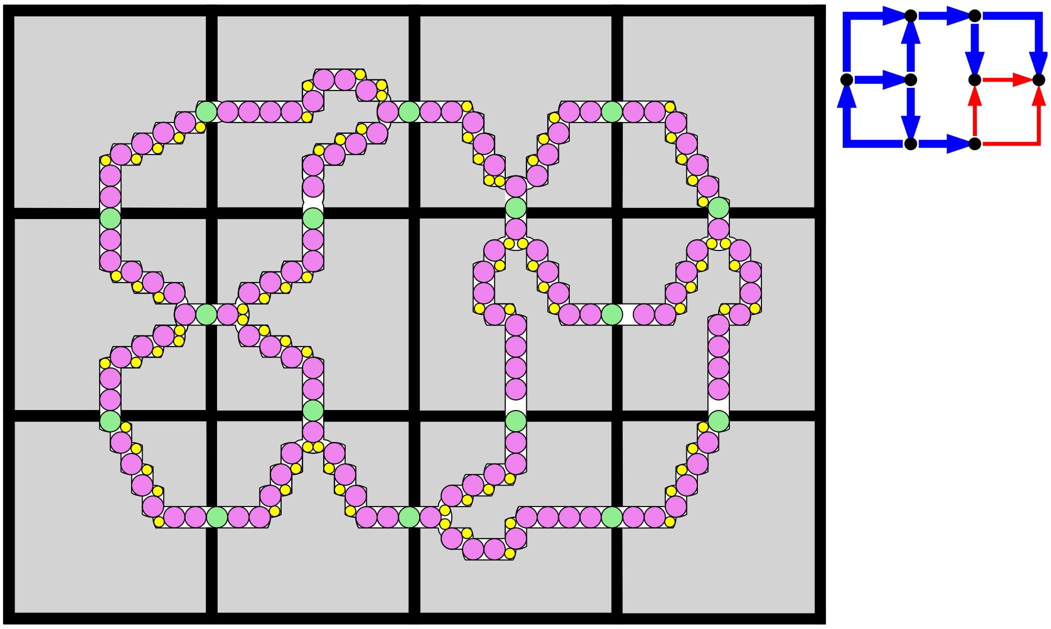


Figure 12 An instance of k-DRMP built for the NCL constraint graph G from Figure 3.

Using the three described types of gadgets, we now build a complete 2-DRMP instance corresponding to the NCL machine on a constraint graph G. We fill the cells of the grid-like environment, dual to an orthogonal drawing of G, with the *AND*, *protected OR*, and connector gadgets. Figure 12 shows an example of a 2-DRMP instance constructed for the constraint graph from Figure 3. The construction of the 2-DRMP instance, and Lemmas 3, 4, and 5, imply the main results of this paper, which we formally prove in the next section.

▶ **Theorem 6.** *The multi-to-multi, multi-to-single, and multi-to-single-in-class k-DRMP problems are PSPACE-hard for two classes of unlabeled disc robots moving amidst obstacles constructed out of line segments and circular arcs.*

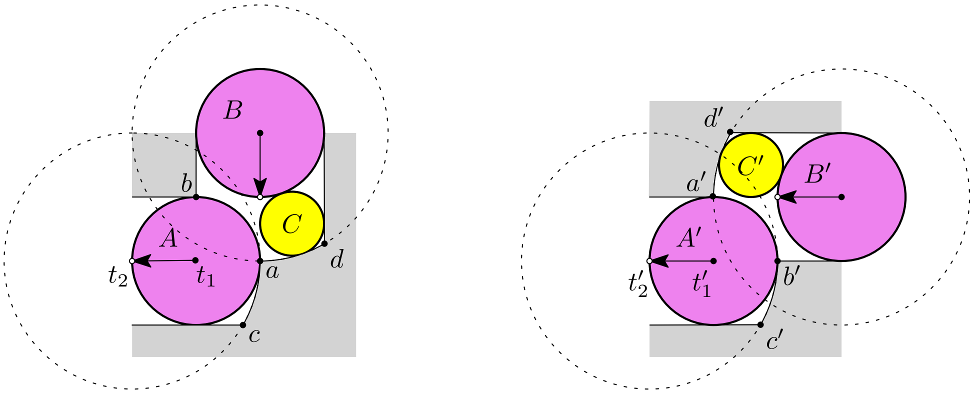
Finally, we argue that the circular arcs in the construction can be approximated with circumscribed polygonal chains without changing the validity of the reduction.

▶ **Theorem 7.** *The multi-to-multi, multi-to-single, and multi-to-single-in-class k-DRMP problems are PSPACE-hard for two classes of unlabeled disc robots moving in a polygonal environment.*

3.2 Correctness of the gadgets

We now prove that the gadgets described in the previous section indeed correspond to their respective nodes in a constraint graph. Let us first take a closer look at the parallel and perpendicular components that make up our gadgets.

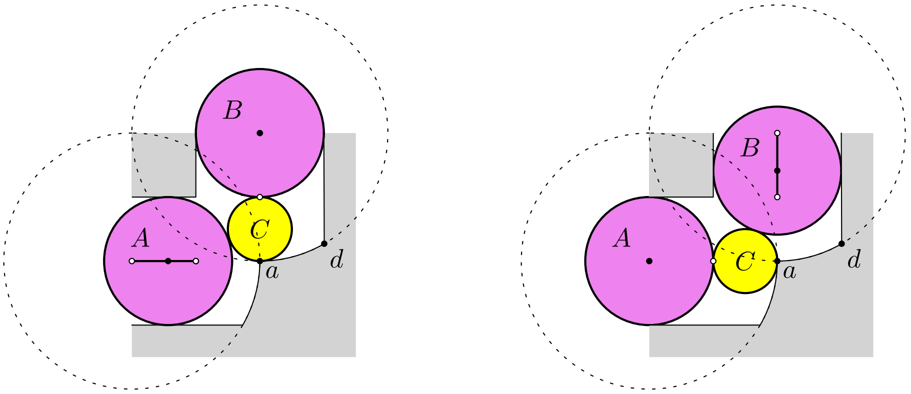
Properties of parallel and perpendicular components. Figure 13 shows a detailed design of the parallel (left) and perpendicular (right) components. Discs A and B have radius 1, disc C has radius 1/2, and the dashed circles have radius 2. These configurations can be mirrored and rotated by 90°, 180° and 270°. We will chain the components to enforce the terminal positions of the edge robots. The red arrows indicate movement of the unit discs in the sketched situation. The dashed circles indicate the circular segments of the boundary of the free space. The white dots indicate terminal positions of A and B.



(a) Perpendicular component.

(b) Parallel component.

Figure 13 Detailed design of the perpendicular (a) and parallel (b) components which are present in every gadget. The two possible components of corner situations inside a gadget.



(a) Small disc on the corner and on the same axis as B.

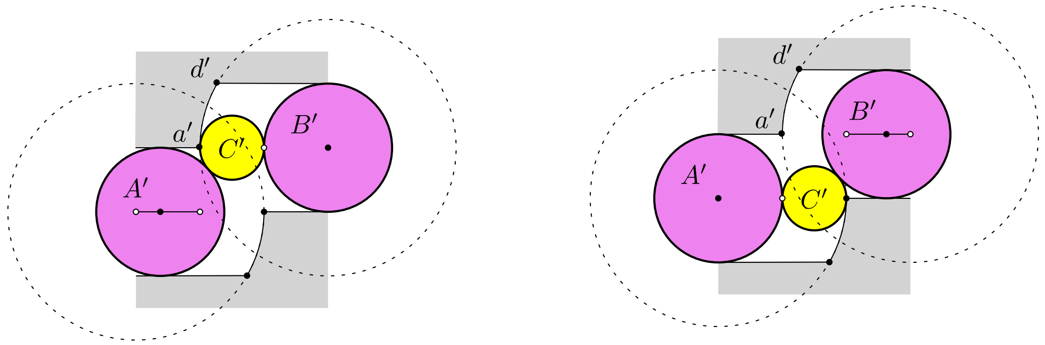
(b) Small disc on the corner and on the same axis as A.

Figure 14 Perpendicular movement with the discs shifted.

We first look at the definition of the perpendicular component as shown in Figure 13 (left). Let point a be at $(0, 0)$. Then points $b = (-1, 1)$, $c = (\sqrt{3}-2, -1)$, and $d = (1, 2-\sqrt{3})$. Points $t_1 = (-1, 0)$ and $t_2 = (-2, 0)$ are terminal positions of disc A. Indeed, by construction the center of disc A must lie on the closed segment t_1t_2. In the configuration of the discs depicted in the figure, disc A is centered at t_1, disc B is centered at $(0, 2)$, and disc C is centered at $(0.5, 2 - \sqrt{2})$. The circular segments ac and ad are arcs of 30° of radius-2 circles centered at t_2 and $(0, 2)$ respectively.

Now we define the parallel component as shown in Figure 13 (right). Let point a' be at $(0, 0)$. Then $b' = (1, -1)$, $c' = (\sqrt{3} - 1, -2)$, and $d' = (2 - \sqrt{3}, 1)$. Points $t'_1 = (0, -1)$ and $t'_2 = (-1, -1)$ are terminal positions of disc A'. In the example depicted in the figure, disc A' is centered at t'_1, disc B' is centered at $(2, 0)$, and disc C' is centered at $(2 - \sqrt{2}, 0.5)$. The circular segments $b'c'$ and $a'd'$ are arcs of 30° of the radius-2 circles centered at t'_2 and $(2, 0)$ respectively.

▶ **Lemma 8.** *In the perpendicular component, disc B can not move down before disc A has moved by distance of at least $\sqrt{2} - 1$ from t_1 towards t_2.*



(a) Small disc on the corner and on the same axis as B.

(b) Small disc on the corner and on the same axis as A

Figure 15 Perpendicular movement with the discs shifted.

Proof. Consider the perpendicular configuration depicted in Figure 13a. Since the arc ad belongs to a circle of radius 2, B can not move before the x-coordinate of the center of C is smaller than the x-coordinate of a. When the x-coordinate of the center of C is equal to the x-coordinate of a, B is still in its topmost position, but A must have moved by at least $\sqrt{2} - 1$ from t_1 to prevent overlap with C (see Figure 14a). ◀

▶ **Lemma 9.** *In the parallel component, disc B' can not move left before disc A' has moved by a distance at least $\frac{\sqrt{5}}{2} - \frac{1}{2}$ from t'_1 towards t'_2.*

Proof. Consider the parallel configuration depicted in Figure 13b. Since the arc $a'd'$ belongs to a circle of radius 2, B' can not move before the y-coordinate of C' becomes smaller than the y-coordinate of a'. When the y-coordinate of C' is equal to the y-coordinate of a', B' has not been able to move yet, but A' must have moved by at least $\frac{\sqrt{5}}{2} - \frac{1}{2}$ to prevent overlap with C' (see Figure 15a). ◀

Proof of Lemma 2. The proof directly follows from Lemmas 8 and 9. Consider, as an example a chain of a perpendicular and a parallel component as depicted in Figure 16. By Lemma 9, disc B' can only move away from its terminal position t'_2 when the radius-(1/2) disc C' moves left beyond point a'. Thus, disc B has to move down by at least $\frac{\sqrt{5}}{2} - \frac{1}{2}$ before B' can move. However, if disc B moves $2 - \sqrt{2}$ or more, by Lemma 8, A must be in the terminal position t_2. As $2 - \sqrt{2} < \frac{\sqrt{5}}{2} - \frac{1}{2}$, we have that either disc B' must be in its terminal position t'_2, or disc A must be in its terminal position t_2. If the chain of parallel and perpendicular components is longer, the extremal positions of the radius-1 discs beyond either A or B' are enforced. Thus, if there is a perpendicular and a parallel component in the chain, the lemma holds. ◀

Correctness of connector gadgets. The correctness of the connector gadgets follows immediately from Lemma 2.

Proof of Lemma 5. As the connector gadgets consist of chains containing a parallel and a perpendicular component, by Lemma 2, the current lemma holds. ◀

Correctness of AND gadgets. The *AND* gadgets, as shown in Figure 9, use two small disc robots with a radius of 1/2. They have a specific layout of the obstacles, which enable the functionality of the gadgets. A close-up view of this layout can be seen in Figure 17. Disc

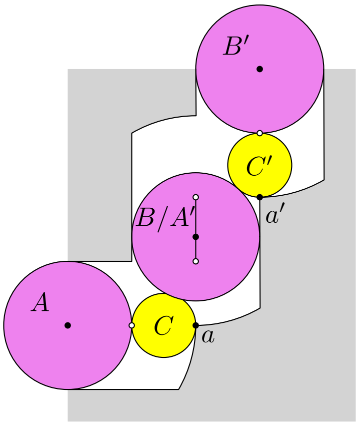


Figure 16 A parallel and a perpendicular component chained. If disc C is horizontally aligned with a, and disc C' is vertically aligned with a', then disc B must overlap either C or C'.

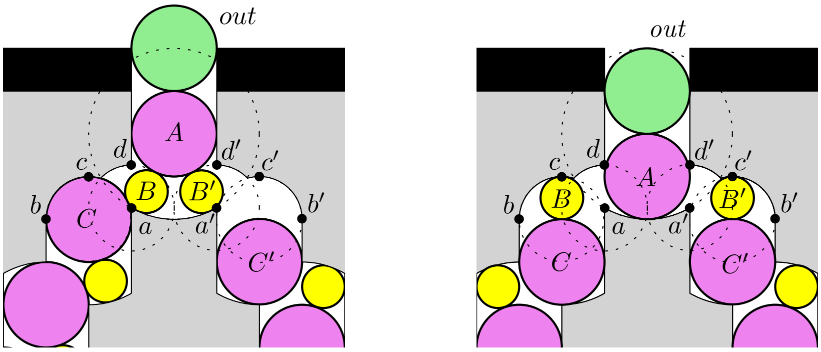


Figure 17 A close-up view of the functionality of the *AND* gadgets, the full gadgets can be seen in Figure 9.

robots B and B' can move along the arc $a'a$ which lies on a circle of radius 2. This means that A can not move as long as either disc B or disc B' is on the arc $a'a$. Arcs cd, $c'd'$, bc, and $b'c'$ lie on circles of radius 1. Thus discs C and C' tightly fit along the respective arcs bc and $b'c'$. Points c and c' are terminal position of the discs B and B'. Indeed, they cannot move further than c (or c'), otherwise they would overlap with disc C (or C'). Thus, discs B and B' cannot both fit in the free space above C or in the free space above C'. Therefore, both discs C and C' must move down to enable disc A to move down. If A is able to reach the arc aa', the discs B and B' should have x-coordinates at most $a - 1/2$ and at least $a' + 1/2$ respectively. Then, discs C and C' must move down by at least distance 0.8539. A similar argument to the one in Lemmas 8 and 9 will force the radius-1 discs to be moved to their terminal positions in the direction away from A.

Proof of Lemma 3. In Figure 17 the *AND* gadget is depicted in more detail. Since the construction is similar to the connector gadget, we know by Lemma 5 that discs C and C' from Figure 17 can only move when the input edge robots are in their terminal configurations. By the described construction above, disc A can only move if both C and C' have moved down. Since C and C' can only move when the inputs are in a terminal configuration, the edge robot *out* can only move when both edge robots in_1 and in_2 are in their outside terminal configuration. ◀

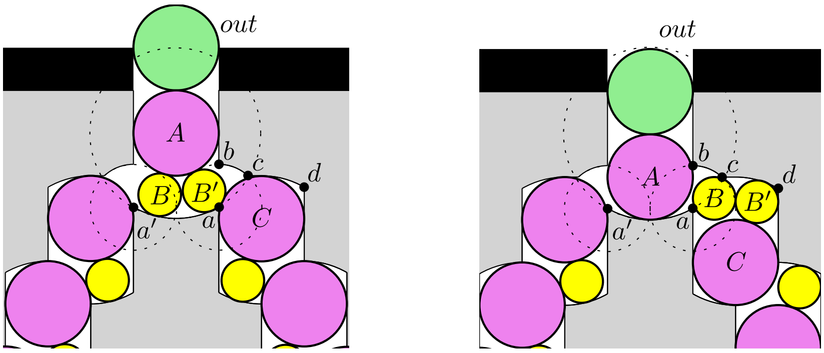


Figure 18 A close-up view of the functionality of the *protected OR* gadget, the full gadget can be seen in Figure 10.

Correctness of protected OR gadgets. The *protected OR* gadgets, as shown in Figure 10, use two small disc robots with a radius of 1/2, just like the *AND* gadgets. However, unlike in the *AND* gadgets, the *protected OR* gadgets have a different layout of the obstacles, which changes the functionality of the gadgets. A close-up view of this layout can be seen in Figure 18. Disc robots B and B' can move along the arc $a'a$ which lies on a circle of radius 2. As long as B or B' lies on this arc, disc A can not move. Disc C might move down, which makes space for discs B and B' to move above C. The arc bc lies on a circle of radius 1. Discs B and B' have a radius of 1/2, so they tightly fit along arc bc when C has made space. Since the arc cd lies on a circle of radius 2, both B and B' fit above C. When both B and B' have moved into one of the sides, A can move down such that the edge disc can move inside the gadget. In our construction we forbid the two robots B and B' to separate and move into the free space pockets above the two different radius-1 discs. We can do so because, as we will argue later, for any valid reconfiguration of robots that separates the discs, there will be an equivalent reconfiguration which keeps the discs always together.

Proof of Lemma 4. The proof directly follows from Lemma 2. ◀

Observe that, after putting all the gadgets together, all edge robots have a limited set of movements and that inner gadget robots remain in their gadgets.

▶ **Observation 10.** *Each edge robot can be in at most two distinct terminal configurations.*

3.3 Reduction

We are now ready to prove our main results.

▶ **Theorem 6.** *The multi-to-multi, multi-to-single, and multi-to-single-in-class k-DRMP problems are PSPACE-hard for two classes of unlabeled disc robots moving amidst obstacles constructed out of line segments and circular arcs.*

Proof. For a given NCL machine built on a constraint graph G, consider the corresponding instance of the k-DRMP problem constructed as described above. Furthermore, consider a state σ of the NCL machine, and a corresponding 2-configuration $\mathcal{S}$. The positions of the edge robots in $\mathcal{S}$ correspond to the orientation of the respective edges of G in σ. By Lemmas 3, 4, and 5, if an edge $e \in G$ cannot be flipped, the corresponding inner-gadget radius-1 robots are forced to be in one of their terminal positions, and cannot move. If, however, e can be flipped, some of the corresponding radius-1 robots are able to move between their terminal positions (refer to Figure 12 for an example). By Lemmas 3, 4, and 5, a flip of the edge e is valid if and only if the corresponding edge robot can move to the opposite terminal position.

Consider the first problem, the *multi-to-multi* k-DRMP. We will show that it is PSPACE-hard by a reduction from the *state-to-state* NCL problem. Recall that the *state-to-state* NCL problem asks whether for a given constraint graph G and for two valid states σ_1 and σ_2 of the NCL machine, σ_1 can be transformed into σ_2 with edge-flip operations. From $\mathcal{S}$ and $\mathcal{T}$ we construct two 2-configurations $\mathcal{S}'$ and $\mathcal{T}'$, such that all the edge robots are in the same positions as in $\mathcal{S}$ and $\mathcal{T}$, all inner radius-1 and radius-(1/2) robots are shifted to their terminal positions consistent with the orientation of the corresponding edges in G. We claim that σ_1 can be transformed into σ_2 with edge-flip operations if and only if the robots can be reconfigured from $\mathcal{S}'$ to $\mathcal{T}'$.

Assume that there is a sequence of edge flips transforming σ_1 into σ_2. For each flip, by Lemmas 3, 4, and 5, we can reconfigure the robots of the k-DRMP instance in correspondence to the changes of orientations of the flipped edges.

It remains to show that if the robots of the k-DRMP instance can be reconfigured from $\mathcal{S}'$ to $\mathcal{T}'$, then there is a valid edge-flip sequence transforming σ_1 into σ_2. Consider the reconfiguration over time, and extract the order in which the edge robots reach one of their terminal configurations. If two edge robots are both in some intermediate positions between their terminal locations, then these edge robots can move independently from one another. We can modify the reconfiguration schedule such that at each moment in time only one edge robot can be located at an intermediate position between its terminal positions.

We still need to argue that we can preserve the "protected" property of the *protected OR* gadgets. Suppose that in the process of reconfiguration, at some moment, two input edge robots are moved to the outside terminal positions. If one of them, say in_1, reverts before the robot out moves to its inner terminal position, then we simply ignore the move of in_1 (and the robots in the chain from in_1 to out) outside. Let robot out move to the inner terminal position. Consider the positions of the discs B and B' (recall Figure 18). If they both are located above one radius-1 disc in the chain from in_1 to out, then we can change the schedule to stop in_2 from moving to the outside terminal position. If B and B' are separated, then we can modify the schedule to move B and B' into the same free-space pocket, and stop the other input edge robot from moving outside. In all cases, we can modify the schedule to prevent both input edge robots to be in their outside terminal positions. Thus, we have a reconfiguration schedule which preserves the "protected" property of the *protected OR* nodes, and has edge robots move between their terminal positions one at a time.

The order in which the edge robots move between their terminal positions gives the order of valid edge flips in G. Indeed, by Lemmas 3, 4, and 5, if an edge robot, corresponding to some edge $e = (u, v) \in G$, can move, the in-flow property of the two corresponding nodes u and v in G is satisfied by the edges other than e. Thus, the multi-to-multi k-DRMP problem is PSPACE-hard.

Now, consider the multi-to-single and multi-to-single-in-class versions of the k-DRMP problem. By a reduction from state-to-edge NCL problem, we show that these two problems are PSPACE-hard. The argument follows the same lines as for the multi-to-multi case, except that instead of the target 2-configuration $\mathcal{T}$, we are given a target location for a robot. We will select the edge robot corresponding to the edge to be flipped in the NCL problem, and specify a proper terminal positions as the target location for the robot. ◀

Remark. Note, that we can remove the use of circular arcs in our construction. Consider a small fixed value $\varepsilon > 0$. There exists a value $d(\varepsilon) > 0$, such that, if we replace the arcs in the construction with circumscribed polygonal chains with edge length at most d, the edge robots will be bound to ε-neighborhoods of the terminal positions. Indeed, for small enough ε, Lemmas 3, 4, and 5 will still hold, with modified statements considering the ε-neighborhoods of the terminal positions instead of simply the terminal positions. Thus, the following result holds.

▶ **Theorem 7.** *The multi-to-multi, multi-to-single, and multi-to-single-in-class k-DRMP problems are PSPACE-hard for two classes of unlabeled disc robots moving in a polygonal environment.*

4 Conclusion

In this paper we have shown that the three variants of the disc-robot motion planning problem are PSPACE-hard, even for two classes of unlabeled disc robots with two different radii, moving in a polygonal environment. This is a first step towards settling the complexity of unlabeled unit disc robot motion planning. Our gadgets do not seem to generalize to a single class of robots. The complexity of unlabeled unit disc robot motion planning remains an interesting open problem.

References

1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *IEEE Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015. doi:10.1109/TASE.2015.2470096.

2 Tiziana Calamoneri and Rossella Petreschi. An efficient orthogonal grid drawing algorithm for cubic graphs. In *International Computing and Combinatorics Conference (COCOON)*, LNCS, volume 959, pages 31–40. 1995. doi:10.1007/BFb0030817.

3 Gary W. Flake and Eric B. Baum. Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1-2):895–911, 2002. doi:10.1016/S0304-3975(01)00173-6.

4 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.

5 John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's Problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

6 Kiril Solovey and Dan Halperin. k-Color multi-robot motion planning. *The International Journal of Robotics Research*, 33(1):82–97, 2014. doi:10.1177/0278364913506268.

7 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *The International Journal of Robotics Research*, 35(14):1750–1759, 2016.

8 Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion planning for unlabeled discs with optimality guarantees. In *Robotics: Science and Systems XI*. Robotics: Science and Systems Foundation, 2015. doi:10.15607/RSS.2015.XI.011.

9 Paul Spirakis and Chee K. Yap. Strong NP-hardness of moving many discs. *Information Processing Letters*, 19(1):55–59, 1984.

10 John Tromp and Rudi Cilibrasi. Limits of rush hour logic complexity. Manuscript, 2005. URL: http://arxiv.org/abs/cs/0502068.

Efficient Algorithm for Multiplication of Numbers in Zeckendorf Representation

Tomasz Idziaszek
Independent Researcher, Poland
http://algonotes.com
tomasz@algonotes.com

Abstract

In the Zeckendorf representation an integer is expressed as a sum of Fibonacci numbers in which no two are consecutive. We show $O(n \log n)$ algorithm for multiplication of two n-digit numbers in Zeckendorf representation.

For this purpose we investigate a relationship between the numeral system using Zeckendorf representations and the golden ratio numeral system. We also show $O(n)$ algorithms for converting numbers between these systems.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Fibonacci numbers, Zeckendorf representation, multiplication algorithm, Fast Fourier Transform, golden ratio numeral system, Lucas numbers

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.16

1 Introduction

Zeckendorf [12] showed that each non-negative integer has a unique representation as a sum of Fibonacci numbers in which no two consecutive Fibonacci numbers occur. This observation leads to a numeral system. One of its applications is in self-delimiting codes [2], but we can research this system for its own sake as a mathematical curiosity.

A natural question for a numeral system is how can we perform arithmetic operations on numbers in such a system, and how fast can we do it. It was shown that addition and subtraction of n-digit numbers in the Zeckendorf system can be performed in $O(n)$ time [1], so as fast as in the binary system.

But multiplication seems to be much harder. It is straightforward to directly utilize addition and construct a grade-school-like $O(n^2)$ multiplication algorithm for the Zeckendorf system, but so far no one presented a faster method. However, for the binary system we can devise $O(n \log n)$ algorithms [9], by observing that multiplication can be reduced to a convolution (which can be calculated using the Fast Fourier Transform) followed by a normalization phase (carry propagation).

The purpose of this paper is to present $O(n \log n)$ algorithm for multiplication of two n-digit numbers in the Zeckendorf system. The main idea is to reduce it to a convolution followed by a certain normalization phase. The normalization phase can be reduced to $O(\log n)$ additions.

For convolution to work, we use the fact that the Zeckendorf system is closely related to a non-integer positional numeral system that uses the golden ratio as its base. Since this system is positional, convolution can be performed exactly the same as in the binary system.

We complete the algorithm by showing $O(n)$ time procedures for converting between the Zeckendorf system and the golden ratio numeral system.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 16; pp. 16:1–16:9
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Preliminaries

Fibonacci numbers and Zeckendorf representation. The sequence of Fibonacci numbers is defined as follows (see also Table 1 in the appendix):

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}. \tag{1}$$

Note that this definition works for all integers n (even negative ones). In particular we have

$$F_{-n} = (-1)^{n+1} F_n. \tag{2}$$

We can express non-negative integers as weighted sums of Fibonacci numbers. A sequence of integer weights a_i with only a finite number of non-zero weights represents number

$$x = \sum_i a_i F_i.$$

Zeckendorf [12, 6] showed that under the following conditions such a representation (called *the Zeckendorf representation*) is unique:

(Z1) Each a_i is either 0 or 1.
(Z2) There are no adjacent non-zero weights, thus $a_{i+1}a_i = 0$ for all i.
(Z3) $a_i = 0$ for $i < 2$.

See Table 2 for the Zeckendorf representations of numbers from 1 to 30. Note that we write sequences of weights in such a way that the most significant weight is on the left.

There are several ways of showing that every number has a Zeckendorf representation, but it is fruitful for us to recall a proof from [8]. For a given non-negative integer x we can start from a trivial representation with only one non-zero weight $a_2 = x$, since $F_2 = 1$. This representation violates condition (Z1). The idea is to transform the representation in a series of steps, where every step locally changes weights, but the represented value stays the same. At the end we would like to obtain a representation that satisfies all three conditions. We call such a series of steps a *normalization procedure.*

Following [8] we show here one example of such a procedure, although not efficient (later in Theorem 2 we show an efficient normalization procedure.) We keep an invariant that condition (Z3) is always satisfied, but conditions (Z1) and (Z2) might be not. Every time condition (Z2) is not satisfied, we find the left-most pair of adjacent non-zero weights and apply the following transformation on it:

$$0\bar{x}\bar{y} \to 1xy, \tag{3}$$

where $\bar{x} = x+1$. The soundness of this transformation follows directly from the recurrence (1), and it never leads to violation of condition (Z3).

If only condition (Z1) is not satisfied, we apply the following transformation on the left-most weight greater than 1:

$$0\bar{\bar{x}}0y \to 1x0\bar{y}. \tag{4}$$

The soundness follows from the following equality valid for all integers n:

$$2F_n = F_n + (F_{n-1} + F_{n-2}) = F_{n+1} + F_{n-2}. \tag{5}$$

The transformation may violate condition (Z3), but only when fixing weight a_2, thus it increases weight a_0 then. But this weight is multiplied by $F_0 = 0$ anyway, so we can safely set it back to 0.

Every time we apply transformations (3)–(4), the sum $\sum_i 2^i a_i$ increases, so since every integer has a finite number of representations with non-negative weights, the process terminates.

The uniqueness of Zeckendorf representation follows from a counting argument (see [6]).

The golden ratio numeral system. Binet's formula provides a closed-form solution for Fibonacci numbers and shows a relationship between them and the golden ratio $\varphi = \frac{1+\sqrt{5}}{2}$:

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}. \tag{6}$$

In fact, another numeral system closely related to Fibonacci numbers is a positional numeral system using the golden ratio as the base [3, 7]. In this system a sequence of integer weights c_i represents number

$$x = \sum_i c_i \varphi^i.$$

This representation is unique (and we call it *the base-φ representation*; see also Table 2) if it satisfies conditions (Z1) and (Z2). To prove that every non-negative integer has a base-φ representation, observe that φ^i satisfies the same recurrence as Fibonacci numbers F_i, namely

$$\varphi^n = \varphi^{n-1} + \varphi^{n-2}.$$

It means that transformations (3)–(4) also work for base-φ representations, and starting from a trivial representation (with one non-zero weight $c_0 = x$, since $\varphi^0 = 1$) we can use almost the same normalization procedure as before to obtain a representation satisfying conditions (Z1) and (Z2). This time, however, we do not make fixes for condition (Z3), thus non-zero weights can occur also for negative indices.

Addition and subtraction algorithms. Arithmetic operations on Zeckendorf representations were discussed in [5, 4, 10] but authors of these papers either did not analyze time complexity of their algorithms or they provided very weak bounds. The optimal linear-time bounds for addition and subtraction algorithms were given in [1], and we recall them here, since they are needed for the multiplication algorithm presented in this paper.

▶ **Theorem 1.** *There are $O(n)$ algorithms for addition and subtraction of two n-digit numbers in Zeckendorf representation and in base-φ representation.*

Proof. We present here only a method behind the algorithms; the full algorithms with the proofs can be found in [1]. The addition algorithm starts by independently adding weights from corresponding positions in the Zeckendorf (or base-φ) representations, obtaining a sequence of weights from set $\{0, 1, 2\}$. This sequence can violate condition (Z2) by having consecutive 1s and/or condition (Z1) by having 2s (but only adjacent to 0s). Since we need to normalize the sequence, we could apply the transformations (3)–(4), but this would be inefficient. To perform normalization procedure in $O(n)$ we need to be more careful. Let r be the position of the right-most non-zero weight in the sequence.

We move a 4-position-wide window from left to right, applying the following transformations were applicable:

$$020x \to 100\bar{x}, \qquad 030x \to 110\bar{x}, \qquad 021x \to 110x, \qquad 012x \to 101x,$$

where $x \in \{0, 1, 2\}$ and $\bar{x} = x + 1$. These transformations may introduce additional weight 3, but at the end all 3s as well as 2s are removed, restoring condition (Z1). Then we move a 3-position-wide window from left to right and apply transformation 011 → 100, which removes groups of consecutive 1s that are longer than two positions. Finally, we move the same window from right to left, removing remaining adjacent 1s and fully restoring condition (Z2).

That finishes normalization procedure for base-φ representations. Note that in the output sequence the right-most non-zero weight is at position not smaller than $r - 2$.

For Zeckendorf representations we must take some additional local step to fix condition (Z3), but its a technical detail.

For subtraction we run a similar algorithm, but now weights are from set $\{-1, 0, 1\}$, and there are no consecutive 1s nor consecutive -1s. We move a 4-position-wide window from left to right with transformations:

$$x00 \to \bar{x}11, \qquad x\bar{1}0 \to \bar{x}01, \qquad x\bar{1}1 \to \bar{x}02, \qquad x0\bar{1} \to \bar{x}10,$$

where $x \in \{1, 2\}$, $\bar{x} = x - 1$ and $\bar{1} = -1$. The transformations keep a positive weight in the window and use it to cancel any -1s. They may introduce additional 2s, but these are adjacent only to 0s. So at the end we can finish the algorithm by performing normalization procedure for addition algorithm. ◀

The subtraction algorithm can be used to prove the uniqueness of the base-φ representations. Suppose that we have two different sequences of weights c_i and c'_i that satisfy conditions (Z1) and (Z2), and they represent the same value x. Assume that c_i is lexicographically larger than c'_i. If we subtract c'_i from c_i, we get a non-zero sequence of weights satisfying conditions (Z1)–(Z2) that represents value 0. That is impossible, since all φ^i are positive.

3 Multiplication Algorithms

Several multiplication algorithms for Zeckendorf representation were discussed in [1], but the authors did not find any algorithm with time complexity better than $O(n^2)$. They posed this as a "challenging open problem".

The idea of the algorithm presented in this paper is that since base-φ is a positional numeral system, the multiplication in this system can be reduced to a convolution after which we need to do a normalization phase. Such a convolution can be calculated in $O(n \log n)$ time using the Fast Fourier Transform [9].

Let sequences c_j and c'_j be the base-φ representations of numbers x and x'. As a result of the convolution we obtain a sequence of non-negative weights C_i such that

$$C_i = \sum_j c_j c'_{i-j},$$

and the sum $\sum_i C_i \varphi^i$ is equal to the product $x \cdot x'$. The weights C_i can be up to $O(n)$. Linear-time normalization procedures used in addition and subtraction algorithms utilised the fact that weights were of constant size. However, we can devise an efficient normalization procedure for arbitrary weights:

▶ **Theorem 2.** *There is $O(n \log M)$ algorithm that, given a sequence of n integer weights from range $[0, M]$, normalizes it so it satisfy conditions (Z1) and (Z2).*

Proof. Each weight is a number of $m = \lfloor \log M \rfloor + 1$ bits. We create m binary sequences $x_0, x_1, \dots, x_{m-1}$: each number in sequence x_i equals to the i-th bit of the corresponding number in the original sequence. We initialize the answer to be a sequence of 0s.

Then we perform m phases. In phase i (for $i = m-1, \dots, 1, 0$) we multiply the answer by 2 (by executing addition algorithm in $O(n)$ time) and then we add to the answer sequence x_i (again by executing addition algorithm). ◀

Using the above theorem directly, we get a normalization procedure for base-φ representation that works in $O(n \log n)$ time and completes the multiplication algorithm for the golden ratio system.

The idea for a multiplication algorithm for Zeckendorf representation is to convert both numbers into base-φ representation, perform multiplication in base-φ, and then convert the result back to Zeckendorf representation. In the next section we show that such conversions are possible in $O(n)$ time, thus we get the theorem:

▶ **Theorem 3.** *There are $O(n \log n)$ algorithms for multiplication of two n-digit numbers in Zeckendorf representation and in base-φ representation.*

4 Conversions Between Representations

Compare the proofs that every number x has the Zeckendorf and the base-φ representation. In both of them we started with a trivial representation ($a_2 = x$ and $c_0 = x$, respectively), and we performed a certain normalization procedure.

Now imagine that we start from $\alpha_2 = x$ (like in the proof for Zeckendorf representation) and we perform the normalization procedure that ignores condition (Z3) (like in the proof for base-φ representation). We get a sequence of weights α_i that satisfies conditions (Z1)–(Z2) and represents number $x = \sum_i \alpha_i F_i$. But exactly the same procedure applied to initial condition $c_0 = x$ produces the base-φ representation $x = \sum_i c_i \varphi^i$. Therefore $\alpha_i = c_{i-2}$ for all integers i.

Thus the weights of the base-φ representation shifted by two places and applied to Fibonacci numbers yield the same value:

$$\sum_i c_i \varphi^i = x = \sum_i c_{i-2} F_i.$$

Thanks to that property conversion from base-φ to Zeckendorf is easy:

▶ **Theorem 4.** *There is $O(n)$ algorithm for converting an n-digit number in base-φ representation to Zeckendorf representation.*

Proof. We have the base-φ representation $x = \sum_i c_i \varphi^i$. For easier notation denote $\alpha_i = c_{i-2}$. We create four sequences that partition the weights into four parts: for positive indices, for indices close to the origin, for odd negative indices, and for even negative indices:

$$x_{pos} = \sum_{i \ge 2} \alpha_i F_i, \qquad x_{odd} = \sum_{i \ge 2,\ i \text{ odd}} \alpha_{-i} F_i,$$

$$x_{orig} = \alpha_1 \alpha_{-1} F_3 + [\alpha_1 \ne \alpha_{-1}] F_2, \qquad x_{even} = \sum_{i \ge 2,\ i \text{ even}} \alpha_{-i} F_i.$$

Since sequence α_i satisfies conditions (Z1)–(Z2), the above sequences are the Zeckendorf representations of numbers x_{pos}, x_{orig}, x_{odd}, and x_{even}. Moreover, from (2) we know that Fibonacci numbers of negative indices $-i$ are equal to their positive counterparts but with changed sign if i is even, thus $x = x_{pos} + x_{orig} + x_{odd} - x_{even}$. Thus we can calculate the Zeckendorf representation of x using the addition algorithm twice and the subtraction algorithm once. ◀

The representation α_i has been investigated by Zeckendorf who called it a "generalized Fibonacci numeration" [11].

For conversion from Zeckendorf to base-φ we can look at Table 2 that contains the base-φ representations for small integers and observe that certain representations have simple forms, e.g. there are some that contain only two 1s. Moreover, if we allow weights of -1 (we call it *relaxed* base-φ representation) we can get simple representations for some more numbers (see Table 3).

These numbers that happen to have simple forms are called Lucas numbers L_n, and are defined as follows (see also Table 1):

$$L_n = F_{n+1} + F_{n-1}.$$

In particular we have the following relationship between Lucas numbers and the golden ratio, that is easy to check using Binet's formula:

$$L_n = \varphi^n + (-1)^n \varphi^{-n}. \tag{7}$$

Thus it looks like, as Fibonacci numbers were the simplest building blocks of Zeckendorf representations, we could use Lucas numbers as simple building blocks of relaxed base-φ representations. Thus we can try to express integers using weighted sums of Lucas numbers; a sequence of integer weights b_i represents number

$$x = \sum_i b_i L_i.$$

Each non-negative integer can be represented as a weighted sum of Lucas numbers, and this representation is unique (thus we call it *the Lucas representation*, see [6] and Table 2) if it satisfies conditions (Z1), (Z2) and a slightly modified version of condition (Z3):

(Z3′) $a_i = 0$ for $i < 0$, and $a_2 a_0 = 0$.

Such a representation can be obtained almost greedily from the Zeckendorf representation:

▶ **Theorem 5.** *There is $O(n)$ algorithm for converting an n-digit number in Zeckendorf representation to Lucas representation.*

Proof. We have the Zeckendorf representation $x = \sum_i a_i F_i$. Like in the proof of Theorem 1, we move a 4-position-wide window from left to right over sequence a_i and apply the following transformations were applicable:

$$100x \to 011x, \qquad 101x \to 0\bar{0}0x, \qquad 1100 \to 0\bar{0}01, \qquad 1101 \to 01\bar{1}0, \qquad 1110 \to 0\bar{1}00,$$

where $x \in \{0, 1\}$. A bar over a digit at position i means that we must increment weight b_i by one. This way we are zeroing sequence a_i, and at the same time constructing the Lucas representation b_i, keeping the sum $\sum_i (a_i F_i + b_i L_i)$ unchanged. We keep the invariant that all the time sequence a_i satisfies conditions (Z1)–(Z2), except the left-most group of consecutive 1s, which can be of length up to 3. Thanks to that every transformation applied replaces the left-most 1 with 0.

We stop after the transformation over positions 3 through 0. Then we finish by applying one of the following transformations over positions 2 through 0:

$$110 \to 00\bar{0}, \qquad 100 \to 0\bar{0}0, \qquad 010 \to 0\bar{0}0, \qquad 001 \to 000.$$

Each weight b_i can be incremented at most twice (if it is incremented twice, then one increment is due to transformation $1101 \to 01\bar{1}0$ followed by transformation $1100 \to 0\bar{0}01$). Moreover, careful examination of the transformations shows that no two adjacent weights are incremented. Also, there is at most one increment over positions 2 through 0, thus the last three weights are in set $\{000, 001, 010, 100\}$.

Thus we have sequence b_i of weights from set $\{0, 1, 2\}$, and 2s are adjacent to 0s on both sides. Since Lucas numbers also satisfy recurrence $L_n = L_{n-1} + L_{n-2}$, we can perform the same normalization procedure as in addition algorithm for base-φ representation, which would help us restore conditions (Z1) and (Z2). However, this procedure could introduce some non-zero weights b_{r-1} and b_{r-2}, where r is the index of the right-most non-zero weight in b_i; so we must be careful here.

If the last three weights are 000 or 100, we are safe, since $r \geq 2$. For the last three weights being 001 or 010 we decrement weight b_r to zero, perform the normalization procedure, and increment b_r back.

For 001 case we end up either with 0101 (which is safe) or with 011 (which we replace with 100 and perform the normalization once again).

For 010 case we end up either with 0110 (which we replace with 1000 and perform the normalization once again) or with 020 (which we replace with 001).

This restores conditions (Z1) and (Z2). If condition (Z3′) is not satisfied, the representation ends in $x0101$. We can replace it with $x1010$ and if $x = 1$ we run a 3-position-wide window from right to left with transformation $011 \to 100$. ◀

Finally, we show conversion from Lucas to base-φ. Conversion from Zeckendorf to base-φ goes through an intermediate step of the Lucas representation.

▶ **Theorem 6.** *There is $O(n)$ algorithm for converting an n-digit number in Lucas representation to base-φ representation.*

Proof. From the Lucas representation $x = \sum_i b_i L_i$ we can directly obtain a sequence of weights c_i in a relaxed base-φ representation, by using (7):

$$c_i = \begin{cases} b_i & i > 0, \\ 2b_0 & i = 0, \\ (-1)^i b_{-i} & i < 0. \end{cases}$$

If $c_0 = 0$, all weights are from set $\{-1, 0, 1\}$, so we can normalize this sequence by performing the normalization procedure just like in subtraction algorithm for base-φ representation. Otherwise $c_0 = 2$, and first we zero this weight, perform the subtraction normalization, then increment c_1 and c_{-2} by one and perform the normalization procedure like in addition algorithm. ◀

5 Alternative Explanation

It turns out that we can invent the multiplication algorithm presented in this paper without using terminology of the golden ratio numeral system, and staying only in the realm of Fibonacci numbers. In fact this was the way it was invented by the author of this paper. Since the obtained results are the same, and we simply change the language used, this section present only the essence of this approach.

The idea is to begin with the observation that starting from trivial representation $\alpha_2 = x$ and performing normalization procedure that ignores condition (Z3) gives us a sequence α_i satisfying conditions (Z1)–(Z2) and representing number

$$x = \sum_i \alpha_i F_i.$$

But if we start exactly the same procedure from an initial condition $\alpha'_m = x$, which represents number xF_m, we get an equation

$$xF_m = \sum_i \alpha'_i F_i = \sum_i \alpha_{i+2} F_{i+m}. \tag{8}$$

The equation captures an important property that shifting sequence α_i results in multiplying the represented value by consecutive Fibonacci numbers. In positional systems (like the binary or the golden ratio system) an analogous property states that shifting a sequence of digits results in multiplying the represented value by the base (2 or φ, respectively).

This "shiftable" property is enough for convolution to work:

$$x \cdot x' = x\Big(\sum_i a'_i F_i\Big) = \sum_i a'_i x F_i \overset{(8)}{=} \sum_i a'_i \Big(\sum_j \alpha_{j+2} F_{j+i}\Big) = \sum_i \Big(\sum_j a'_{i-j} \alpha_{j+2}\Big) F_i.$$

Thus it is enough to convert one of the arguments to a "shiftable" representation α_i, and perform convolution to obtain weights $A_i = \sum_j a'_{i-j}\alpha_{j+2}$ of a "shiftable" representation of the product.

For conversion part we observe that the representations α_i of Lucas numbers are particularly simple, which results from the following equivalent of equation (7) that holds for any integers n and m:

$$L_n F_m = F_{m+n} + (-1)^n F_{m-n}.$$

References

1 Connor Ahlbach, Jeremy Usatine, Christiane Frougny, and Nicholas Pippenger. Efficient algorithms for Zeckendorf arithmetic. *The Fibonacci Quarterly*, 51(3):249–255, 2013.

2 Alberto Apostolico and Aviezri S. Fraenkel. Robust transmission of unbounded strings using Fibonacci representations. *IEEE Transactions on Information Theory*, 33(2):238–245, 1987.

3 George Bergman. A number system with an irrational base. *Mathematics Magazine*, 31(2):98–110, 1957.

4 Peter Fenwick. Zeckendorf integer arithmetic. *Fibonacci Quarterly*, 41:405–413, 2003.

5 H.T. Freitag and G.M. Phillips. *Elements of Zeckendorf Arithmetic*, pages 129–132. Springer Netherlands, Dordrecht, 1998.

6 Verner E. Hoggatt, Jr. *Fibonacci and Lucas Numbers*. Houghton Mifflin Company, 1969.

7 Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison–Wesley, 1968.

8 Donald E. Knuth. Fibonacci multiplication. *Applied Mathematics Letters*, 1(2):III–VI, 1988.

9 Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

10 Garry J. Tee. Russian peasant multiplication and Egyptian division in Zeckendorf arithmetic. *Australian Mathematical Society Gazette*, 30(5):267–276, 2003.

11 Edouard Zeckendorf. A generalized Fibonacci numeration. *The Fibonacci Quarterly*, 10(4):365–372, 1972.

12 Edouard Zeckendorf. Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. *Bull. Soc. Roy. Sci. Liège*, 41:179–182, 1972.

Table 1 Fibonacci and Lucas numbers.

n	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
F_n	21	13	8	5	3	2	1	1	0	1	-1	2	-3	5	-8	13	-21
L_n	47	29	18	11	7	4	3	1	2								

Table 2 Zeckendorf, base-φ, and Lucas representations.

n	Zeckendorf	base-φ	Lucas
1	100	1.	10
2	1000	10.01	1
3	10000	100.01	100
4	10100	101.01	1000
5	100000	1000.1001	1010
6	100100	1010.0001	1001
7	101000	10000.0001	10000
8	1000000	10001.0001	10010
9	1000100	10010.0101	10001
10	1001000	10100.0101	10100
11	1010000	10101.0101	100000
12	1010100	100000.101001	100010
13	10000000	100010.001001	100001
14	10000100	100100.001001	100100
15	10001000	100101.001001	101000
16	10010000	101000.100001	101010
17	10010100	101010.000001	101001
18	10100000	1000000.000001	1000000
19	10100100	1000001.000001	1000010
20	10101000	1000010.010001	1000001
21	100000000	1000100.010001	1000100
22	100000100	1000101.010001	1001000
23	100001000	1001000.100101	1001010
24	100010000	1001010.000101	1001001
25	100010100	1010000.000101	1010000
26	100100000	1010001.000101	1010010
27	100100100	1010010.010101	1010001
28	100101000	1010100.010101	1010100
29	101000000	1010101.010101	10000000
30	101000100	10000000.10101001	10000010

Table 3 Base-φ representations of Lucas numbers and their simple forms in relaxed base-φ.

n	base-φ	relaxed base-φ
1	1.	$10.\bar{1}$
3	100.01	100.01
4	101.01	$1000.00\bar{1}$
7	10000.0001	10000.0001
11	10101.0101	$100000.0000\bar{1}$
18	1000000.000001	1000000.000001
29	1010101.010101	$10000000.000000\bar{1}$

Foundations for Actively Secure Card-Based Cryptography

Alexander Koch
Karlsruhe Institute of Technology (KIT), Germany
alexander.koch@kit.edu

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de

Abstract

Card-based cryptography, as first proposed by den Boer [4], enables secure multiparty computation using only a deck of playing cards. Many protocols as of yet come with an “honest-but-curious” disclaimer. However, modern cryptography aims to provide security also in the presence of *active attackers* that deviate from the protocol description. In the few places where authors argue for the active security of their protocols, this is done ad-hoc and restricted to the concrete operations needed, often using additional physical tools, such as envelopes or sliding cover boxes. This paper provides the first systematic approach to *active security* in card-based protocols.

The main technical contribution concerns *shuffling* operations. A shuffle randomly permutes the cards according to a well-defined distribution but hides the chosen permutation from the players. We show how the large and natural class of *uniform closed* shuffles, which are shuffles that select a permutation *uniformly* at random from a permutation *group*, can be implemented using only a linear number of helping cards. This ensures that any protocol in the model of Mizuki and Shizuya [17] can be realized in an actively secure fashion, as long as it is secure in this abstract model and restricted to uniform closed shuffles. Uniform closed shuffles are already sufficient for securely computing any circuit [19]. In the process, we develop a more concrete model for card-based cryptographic protocols with two players, which we believe to be of independent interest.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques; Security and privacy → Usability in security and privacy; Theory of computation → Models of computation

Keywords and phrases Card-Based Protocols, Card Shuffling, Secure Multiparty Computation, Active Security, Cryptography without Computers

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.17

Related Version A full version is available at [10], `https://eprint.iacr.org/2017/423`.

Acknowledgements We would like to thank the anonymous reviewers for helpful comments.

1 Introduction

The elegant “five-card trick” of den Boer [4] allows two players – here called Alice and Bob – to compute a logical AND of two private bits, using five playing cards. For instance, if the bit of a player encodes whether they have romantic interest for the other player, the protocol will result in a “yes”-output if and only if there is mutual interest, sparing a party with an unrequited crush the embarrassment of having this information revealed.

More generally, using a deck of playing cards (usually with symbols ♡, ♣), Alice and Bob can jointly compute an arbitrary Boolean function on multiple secret inputs such that neither player learns anything about the input, except, possibly, what can be learned from looking at the output. One distinctive feature is that these protocols do not need a computer, which makes their security tangible. For this reason, they have become popular for introducing secure multiparty computation in lectures and to non-experts.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 17; pp. 17:1–17:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The key operations that introduce randomness in a controlled manner are shuffles. A shuffle operation causes a sequence of cards to be rearranged according to random permutation such that observers cannot tell which permutation was chosen. The formal computational model of Mizuki and Shizuya [17] permits shuffles with arbitrary distributions on permutations. The model is useful when showing impossibility results and *lower bounds* on cards, cf. [12], but it seems unlikely that all shuffle operations permitted in the model have a convincing real world implementation. This spawned some formal protocols with apparently good parameters, but unclear real-world implementations, especially if active security is a concern [12, Sect. 7]. There is to this day still no *positive* account of what shuffles can be done with playing cards beyond the justification of individual protocols, and even then, most make "honest-but-curious" assumptions, with no guarantees when one of the players deviates from the protocol. In several places in the literature, e.g. [2, Sect. 8] and [12, Sect. 9], the need for achieve actively secure shuffles and protocols has been recognized.

Our Contribution

As security guarantees in the physical world are harder to formalize than in the digital domain[1], we *introduce* a suitable notion of active security. It is slightly non-standard in that we exclude attackers that are too strong. For instance, there is no possible defense against attackers that can arbitrarily turn over cards, cf. Section 6 for a discussion. Moreover, we show how any card-based protocol (in the model of [17]) that is restricted to *uniform closed shuffles* can be transformed into an actively secure protocol that increases the number of cards only by a constant factor. Uniform closed shuffles, namely those that rearrange the cards according to a *uniform* distribution on a permutation *group*, have already been identified in [12, Sect. 8] as a natural class of operations. More importantly, they *suffice to compute any function*[2].

Along the way, we define a new model for card-based cryptography, which we call *two-player protocols*. These, in turn, use *permutation protocols* that allow Alice to apply a $\pi \in \Pi$ of her choosing to a sequence of face-down cards, such that Bob learns nothing about her choice. We believe this to be of independent interest, e.g. as an approach to formalize protocols such as the 3-card AND protocol in [13] that does not fit into the model of Mizuki and Shizuya.

The idea of using "private permutations" as base operations instead of shuffles was first mentioned in [12, Sect. 8]. Independently from our work, these operations are used in [23] to more efficiently perform an instance of the millionaires problem with cards and in [22] for the case of a three-input voting protocol. To formalize security correctly however, we have to distinguish between private permutation in the role of introducing uncertainty, and those which should serve as input (and need special protection), which we discuss in Section 8. There, we also discuss active attacks against two majority protocols from the literature, that have their inputs given by the users's choice of permutation.

[1] See also `https://xkcd.com/538/` for a humorous illustration of this fact.

[2] Almost all existing Mizuki–Shizuya protocols, e.g. [3, 4, 6, 15, 14, 16, 19, 18, 25, 24, 31, 1], use only these. This list contains protocols for AND and COPY, hence allowing arbitrary circuits. More general shuffles appear in [2, 12, 29] for the purpose of using less cards. For example, for committed-format AND, restricting to uniform closed shuffles needs exactly one additional card, both in the case of finite runtime and Las Vegas protocols, as shown in [19, 12, 1, 7, 8].

Related Work

The feasibility of general secure multiparty computation with cards was shown in [4, 3, 24, 31]. Since then, researchers proposed a wide range of protocols with different objectives and parameters. One line of research has been to minimize the number of cards used in protocols. In this regard, [19, 16, 12, 28, 7, 1] try to minimize the number of cards for AND, XOR or bit copy protocols, achieving, for instance, the minimum number of four cards for AND protocols both in committed[3] and non-committed format.

With respect to shuffles, all early protocols relied solely on a uniform *random cut*, which is a shuffle causing a cyclic shift on a pile of cards with uniformly random offset. Niemi and Renvall [24, Sect. 3] and den Boer [4] plausibly argue that random cuts can be performed by repeatedly cutting a pile of cards in quick succession, as players are unable to keep track. Other shuffles were justified, including "dihedral group" shuffles [24], [31, Sect. 7], *random bisection cuts* [19, 32] and unequal division shuffles [2, 28, 27].

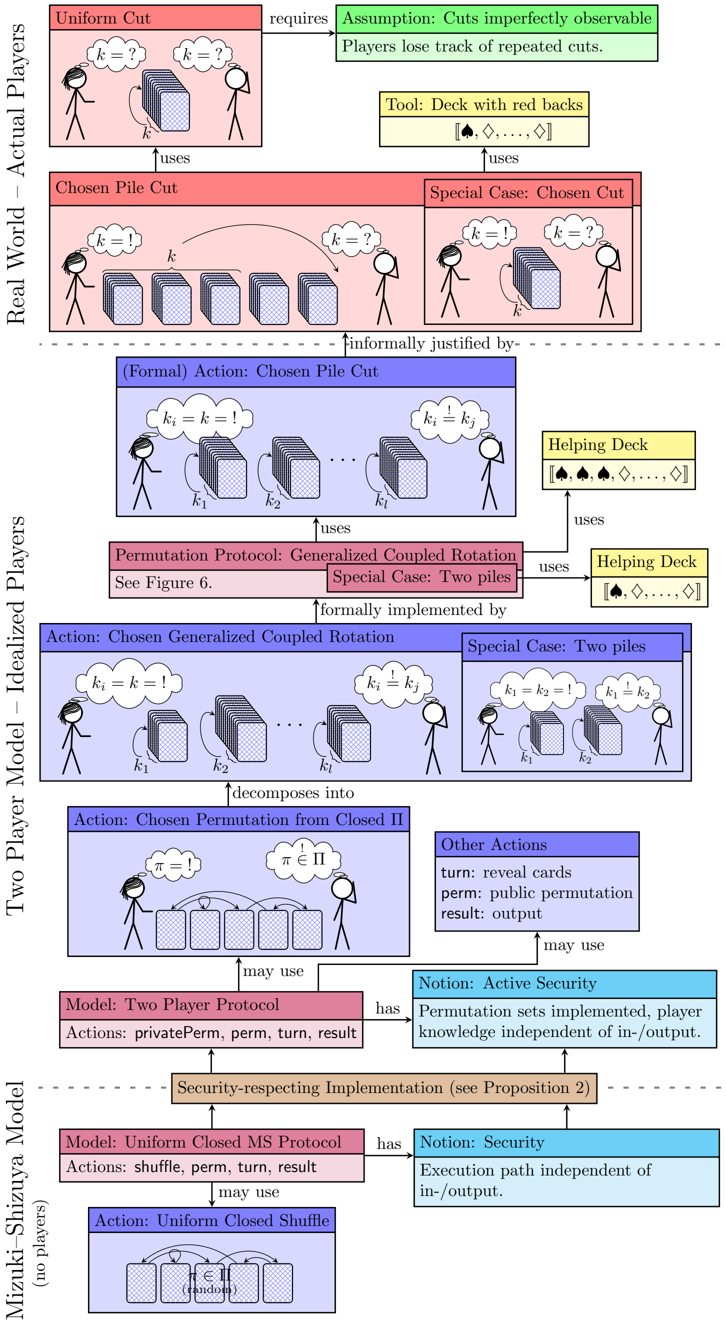
Other works have investigated the question of active attacks, albeit with a different focus. Mizuki and Shizuya [18] address active security against adversaries who deviate from the input encoding, e.g. giving input $(\heartsuit, \heartsuit)$ instead of $(\heartsuit, \clubsuit)$. We describe in Section 8 how our results subsume this, using a separate input phase. Moreover, they stress the necessity of non-symmetric backs to avoid marking cards by rotating them. Finally, using a secret sharing-like mechanism, they specify how to avoid security breaches by scuff marks on the backs of the cards. [30] describe a method against injection attacks in their model using polarizing plates. Independently, [32] give an implementation of the special case of random bisection cuts, including experiments showing the real-world security of the shuffle.

Besides short ad-hoc discussions of the shuffle security, we believe that this is an exhaustive list of all investigations into active security so far. In particular, the issue of ensuring that only permutations allowed in the protocol description can be performed during a shuffle has not been addressed for non-trivial cases. Due to our constructions spanning multiple layers of abstractions as depicted in Figure 1, we are able to solve this by giving a transformation of passively secure protocols into an actively secure ones, under certain conditions.

2 Preliminaries

Permutations. A *permutation* of a set $X = \{1, \dots, n\}$ for some $n \in \mathbb{N}$, is a bijective map $\pi : X \to X$. The set S_n of all permutations of $\{1, \dots, n\}$ is called *symmetric group*. It has group structure with the identity map id as neutral element and composition $(\circ)$ as group operation. We apply a permutation π of X to a set $S \subseteq X$ by writing $\pi(S) := \{\pi(s) \mid s \in S\}$. We say that π *respects* S if $\pi(S) = S$. In that case, π also respects the complement $X \setminus S$ and we can define the *restriction* of π to S as the permutation τ with domain S and $\tau(s) = \pi(s)$ for all $s \in S$. For elements $x_1, \dots, x_k$ the *cycle* $(x_1\ x_2\ \dots\ x_k)$ denotes the *cyclic* permutation π with $\pi(x_i) = x_{i+1}$ for $1 \le i < k$ and $\pi(x_k) = x_1$ and $\pi(x) = x$ for all x not occurring in the cycle. If several cycles act on pairwise disjoint sets, we write them next to one another to denote their composition. For instance $(1\ 2)(3\ 4\ 5)$ denotes a permutation with mappings $\{1 \mapsto 2, 2 \mapsto 1, 3 \mapsto 4, 4 \mapsto 5, 5 \mapsto 3\}$. Every permutation can be written in such a *cycle decomposition*.

[3] In a committed-format protocol, input and output bits are encoded by the order of two face-down cards (a "commitment") that hides the value and hence, may be used as intermediary input to another protocol without looking at it, while those not in committed format reveal the output and are hence unsuitable for larger circuits.


Real World – Actual Players
Uniform Cut
$k = ?$
$k = ?$
k
requires
Assumption: Cuts imperfectly observable
Players lose track of repeated cuts.
Tool: Deck with red backs
$[\![♠, ♢, \ldots, ♢]\!]$
uses
uses
Chosen Pile Cut
$k = !$
k
$k = ?$
Special Case: Chosen Cut
$k = !$
$k = ?$
k
informally justified by
Two Player Model – Idealized Players
(Formal) Action: Chosen Pile Cut
$k_i = k = !$
$k_i \overset{!}{=} k_j$
k_1 k_2 … k_l
Helping Deck
$[\![♠, ♠, ♠, ♢, \ldots, ♢]\!]$
uses
uses
Permutation Protocol: Generalized Coupled Rotation
See Figure 6.
Special Case: Two piles
uses
Helping Deck
$[\![♠, ♢, \ldots, ♢]\!]$
formally implemented by
Action: Chosen Generalized Coupled Rotation
$k_i = k = !$
$k_i \overset{!}{=} k_j$
k_1 k_2 … k_l
Special Case: Two piles
$k_1 = k_2 = !$
$k_1 \overset{!}{=} k_2$
k_1 k_2
decomposes into
Action: Chosen Permutation from Closed Π
$\pi = !$
$\pi \overset{!}{\in} \Pi$
Other Actions
turn: reveal cards
perm: public permutation
result: output
may use
may use
Model: Two Player Protocol
Actions: privatePerm, perm, turn, result
has
Notion: Active Security
Permutation sets implemented, player knowledge independent of in-/output.
Security-respecting Implementation (see Proposition 2)
Mizuki–Shizuya Model
(no players)
Model: Uniform Closed MS Protocol
Actions: shuffle, perm, turn, result
has
Notion: Security
Execution path independent of in-/output.
may use
Action: Uniform Closed Shuffle
$\pi \in \Pi$
(random)

A *uniform cut* (p. 6) rotates a pile of cards by a uniformly random value unknown to Alice and Bob. From this we build *chosen cuts* (p. 6) leaving a pile rotated by a value chosen by Alice but unknown to Bob. When generalized to *chosen pile cuts* (p. 7) and formalized, we obtain a chosen pile cut action that rotates a sequence of equally-sized piles by a value k chosen by Alice. Bob remains oblivious of that value but he can be sure that the cards are not rearranged in any other way. In particular he knows that each pile is rotated by the same amount, even if Alice is dishonest.

With the help of a *permutation protocol* (p. 8) this is extended to the case where piles may have different sizes. This yields *chosen coupled rotations* (p. 8) in the case of two piles and *chosen generalized coupled rotations* (p. 10) in the case of more than two piles.

These are powerful enough to build arbitrary *chosen permutations from a closed permutation set* (p. 10). In that setting, Alice may choose any permutation π from a group of permutations Π. Bob will not learn π but can be sure that no permutation outside the set Π is performed.

A *two player protocol* (p. 11) may make use of these chosen closed permutation actions as well as the *other actions* turn, perm and result.

Uniform closed Mizuki–Shizuya (MS) protocols (p. 16) are a large natural subset of protocols as formalized by Mizuki and Shizuya. Our main result is that for any such protocol there is a two player protocol computing the same function that is *actively secure* (p. 14) if the original protocol is *secure* (p. 13). This *security-respecting implementation* (p. 15) replaces each uniform closed shuffle with two corresponding chosen closed permutations.

Active security is bought with helping cards needed in several places; intuitively to prove the legitimacy of Alice's actions to Bob.

Figure 1 Overview of the content of this paper. The images of Alice and Bob are adapted from xkcd (by Randall Munroe), which is licensed as CC-BY-NC-2.5.

By a *conjugate* of a permutation $\pi \in S_n$ we mean any permutation of the form $\pi' := \tau^{-1} \circ \pi \circ \tau$ where $\tau \in S_n$. For a set $\Pi \subseteq S_n$ of permutations and $\tau \in S_n$ the set $\tau^{-1} \circ \Pi \circ \tau := \{\tau^{-1} \circ \pi \circ \tau \mid \pi \in \Pi\}$ is a *conjugate* of Π. Given an arbitrary sequence of objects $\Gamma = (\Gamma[1], \dots, \Gamma[n])$ and a permutation $\pi \in S_n$ then *applying* π to Γ yields the sequence $\pi(\Gamma) = (\Gamma[\pi^{-1}(1)], \Gamma[\pi^{-1}(2)], \dots, \Gamma[\pi^{-1}(n)])$. Intuitively, the object in position i is transported to position $\pi(i)$.

Sets and Groups. If $g_1, g_2, \dots, g_k \in G$ are group elements, $\langle g_1, \dots, g_k \rangle$ is the smallest subgroup of G containing $g_1, \dots, g_k$ and called the *subgroup generated by* $\{g_1, \dots, g_k\}$. For $g \in G$ the *order* of g is $\mathsf{ord}(g) = |\langle g \rangle| = \min\{k \geq 1 \mid g^k = \mathsf{id}\}$. In the following, a group is implicitly also the set of its elements.

Multisets and Decks. $[\![\diamondsuit, \diamondsuit, \diamondsuit, \spadesuit, \spadesuit]\!]$ is the multiset containing three copies of $\diamondsuit$ and two copies of $\spadesuit$, also written as $[\![3 \cdot \diamondsuit, 2 \cdot \spadesuit]\!]$. If such a multiset represents cards, it is called a *deck*. All cards are implicitly assumed to have the same back, unless stated otherwise. Cards can lie face-up or face-down. When face-down, all cards are indistinguishable (unless they have different backs). When face-up, cards with the same symbol are indistinguishable. Throughout this paper, cards are always face-down with the exception of during a turn operation. To simplify the protocol specification, we immediately turn the card(s) face-down again. Unions of multisets are denoted by $\cup$, disjoint unions are denoted by $+$, e.g. $[\![\diamondsuit, \spadesuit, \spadesuit]\!] \cup [\![\diamondsuit, \heartsuit, \spadesuit, \clubsuit]\!] = [\![\diamondsuit, \heartsuit, \spadesuit, \spadesuit, \clubsuit]\!]$ whereas $[\![\diamondsuit, \spadesuit, \spadesuit]\!] + [\![\diamondsuit, \heartsuit, \spadesuit, \clubsuit]\!] = [\![\diamondsuit, \diamondsuit, \heartsuit, \spadesuit, \spadesuit, \spadesuit, \clubsuit]\!]$.

3 Implementing Cuts and Pile Cuts with Choice

We are interested in procedures that, for a given set $\Pi \subseteq S_n$ of permutations, allow Alice to apply a $\pi \in \Pi$ of her choosing to a sequence of face-down cards, such that Bob learns nothing about her choice, but is certain that Alice did not choose $\pi \notin \Pi$. Also, no player learns anything about the face-down cards if the other player is honest.

In this case we say Π has an *actively secure implementation with choice*, or *is implemented* for short.

Example: Bisection Cut with Envelopes

Mizuki and Sone [19] make use of the following procedure on six cards: The cards in positions 1, 2 and 3 are stacked and put in one envelope and the cards in position 4, 5 and 6 are put into another. Behind her back, Alice then swaps the envelopes or leaves them as they are – her choice. Unpacking yields either the original sequence or the sequence $4, 5, 6, 1, 2, 3$. The bisection cut $\Pi = \{\mathsf{id}, (1\ 4)(2\ 5)(3\ 6)\}$ is therefore implemented (with active security and choice) using two indistinguishable envelopes.

The envelopes ensure that the two groups of cards stay together and their ordering is preserved. The idea is that opening the envelopes behind her back would be impractical and noisy, so even if Alice is malicious, she is limited to the intended options. For a model of secure envelopes, cf. [20, 21].

Example: Unequal Division Shuffle

A bisection cut on n cards can be interpreted as "either do nothing or rotate the sequence by $n/2$ positions". Generalizing this, we now want to "either do nothing or rotate the sequence by l positions" for some $0 < l < n$, i.e. implement $\Pi_l = \{\mathsf{id}, (1\ 2\ \dots\ n)^l\}$. In [28, 29] a corresponding mechanism is described using two card cases with sliding covers. The card cases behave like envelopes but are heavy enough to mask inequalities in weight caused by different numbers of cards, and support joining the content of two card cases – for details refer to their paper (or Appendix D in the full version [10]).

While we are very fond of such creative ideas, in this paper we implement card-based protocols using only one tool: additional cards.

3.1 Cutting the Cards

By the *cut on n cards* we mean the permutation set $\Pi = \langle(1\ \ldots\ n)\rangle$. Alice would *cut* a pile of n cards by taking the top-most k cards (for some $0 < k < n$) from the top of the pile, setting them aside and then placing the remaining $n - k$ cards on top. In this form, Alice can *only approximately* pick k while allowing Bob to approximately observe k. Implementing Π requires fixing both problems.

Uniform Cut

As an intermediate goal we implement a *uniform cut* on n cards, i.e. we perform a permutation $(1\ 2\ \ldots\ n)^k$ for $0 \leq k < n$ chosen uniformly at random and unknown to the players. As proposed in [4], this is done by repeatedly cutting the pile in quick succession until both players lost track of what happened. More formally, under reasonable assumptions, the state of the pile is described by a Markov chain that converges quickly to an almost uniform distribution after a finite number of steps.

Arguably, if the pile is too small, say two cards, the number of cards taken during each cut is perfectly observable. In that case, we put a sufficiently large number c of cards with different backs behind each card, repeatedly cut this larger pile and remove the auxiliary cards afterwards. Note that [32] found it to work well in practice even for $n = 2$ and $c = 3$.[4] We shall not explore this further and use uniform cuts as a primitive in our protocols.

Uniform Cut with Alternating Backs

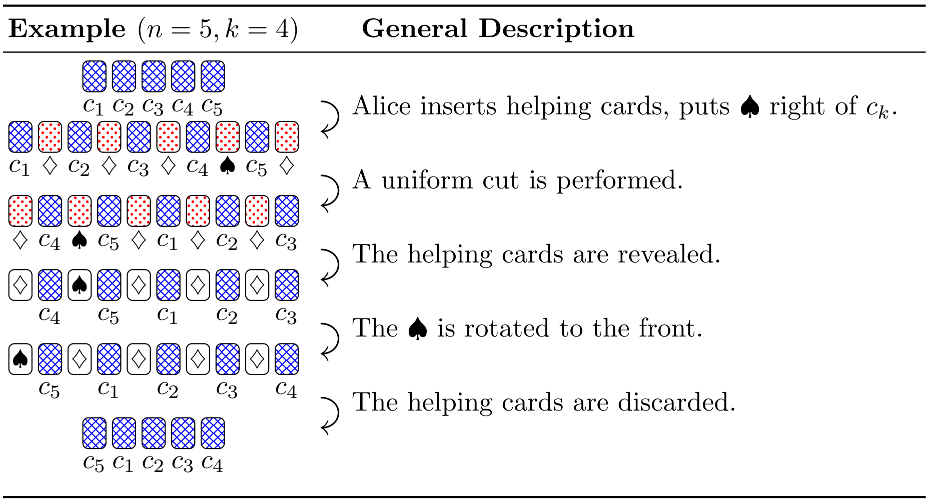
Later we apply the uniform cut procedure to piles of $n \cdot (\ell + 1)$ cards with n cards of red back, each preceded by ℓ cards of blue back. From a “uniform cut” on such a pile, we expect a cut by $0 \leq k < n \cdot (\ell + 1)$ where $\lfloor k/(\ell + 1)\rfloor$ is uniformly distributed in $\{0, ..., n - 1\}$ and independent of the observable part $k \bmod (\ell + 1)$. We leave it to the reader to verify that the iterated cuts still work under the same assumptions.

Chosen Cut

We now show how to implement $\Pi = \langle(1\ \ldots\ n)\rangle$ with active security and choice. Say Alice wants to rotate the pile of n cards by exactly k positions for a secret $0 \leq k < n$. We propose the process illustrated in Figure 2.

Alice is handed the helping deck $[\![\spadesuit, (n-1) \cdot \diamondsuit]\!]$ with red backs and secretly rearranges these cards in her hand, putting $\spadesuit$ in position k. The helping cards are put face-down on the table and interleaved with the pile to be cut (each blue card followed by a red card). The $\spadesuit$ is now to the right of the card that was the k-th card in the beginning. To obscure Alice’s choice of k, we perform a uniform cut on all cards as described previously. The red helping

[4] If not satisfied, the reader may accept some variant of Berry’s turntable, cf. [33].


Example $(n = 5, k = 4)$
General Description
c_1 c_2 c_3 c_4 c_5
c_1 ♢ c_2 ♢ c_3 ♢ c_4 ♠ c_5 ♢
♢ c_4 ♠ c_5 ♢ c_1 ♢ c_2 ♢ c_3
c_4 c_5 c_1 c_2 c_3
c_5 c_1 c_2 c_3 c_4
c_5 c_1 c_2 c_3 c_4

Alice inserts helping cards, puts ♠ right of c_k.

A uniform cut is performed.

The helping cards are revealed.

The ♠ is rotated to the front.

The helping cards are discarded.

Figure 2 Alice cuts a pile of n cards, here $(c_1, \dots, c_5)$, with back ▨ at position k with a helping deck of n helping cards $[\![♠, 4 \cdot ♢]\!]$ with back ░. In this illustration we annotated face-down cards with the symbol they contain.

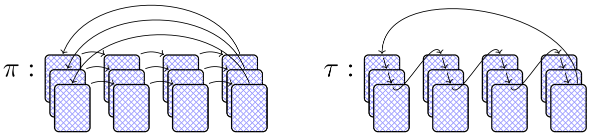


Figure 3 Rotating a sequence of four piles of three cards each by one position *(left)* is described by a permutation π with three cycles of length 4. Alternatively, we can think of π as $\pi = \tau^3$ where τ is the cyclic permutation of length 12 *(right)*.

cards are then turned over. Rotating the sequence so as to put ♠ in front, and removing the helping cards afterward leaves the cards in the desired configuration. Bob is clueless about k since he only observes the position of ♠ *after* the cut, which is independent of the position of ♠ before the cut (which encodes k).

Chosen Pile Cut

Chosen cuts can be generalized in an interesting way. Given n piles of ℓ cards each and $0 \leq k < n$, Alice wants to rotate the sequence of piles by exactly k positions, meaning the i-th pile will end up where pile $i + k$ has been (modulo n). Again, k must remain hidden from Bob and he, on the other hand, wants to be certain that Alice does not tamper with the piles in any other than the stated way. Note that this is equivalent to cutting a pile of $n\ell$ cards where only cutting by multiples of ℓ is allowed, see Figure 3. In that interpretation, the i-th pile is made up of the cards in positions $(i-1)\ell + 1, \dots, i\ell$.

We apply the same procedure as before with n helping cards, except this time, instead of a single blue card we have ℓ blue cards (a pile) before each of the n gaps that Alice may fill with her red deck $[\![♠, (n-1) \cdot ♢]\!]$. Now the special ♠-card marks the end of the k-th pile and is (after a uniform cut) rotated to the beginning of the sequence, ensuring that after removing the helping cards again we end up having rotated the $n \cdot \ell$ cards by a multiple of ℓ as desired. Note that, uniform (non-chosen) pile cuts have been proposed in [6] as "pile-scramble shuffles", with an implementation using rubber bands, clips or envelopes.

Summary

If $\Pi = \langle (1\ 2\ \dots\ n \cdot \ell)^\ell \rangle$ for $n, \ell \in \mathbb{N}$, then Π is implemented with active security and choice using the helping deck $[\![\spadesuit, (n{-}1) \cdot \heartsuit]\!]$. For $\ell = 1$ it is called a *cut*, for $\ell > 1$ a *pile cut*. We use the same name for conjugates of Π, i.e. if cards are relabeled. Any subset $\emptyset \neq \Pi' \subset \Pi$ of a (pile) cut is also implemented: Alice places ♠ only in some positions, the others are publicly filled with ♡.

4 Permutation Protocols for Arbitrary Groups

We introduce a formal concept that allows to compose simple procedures to implement more complicated permutation sets.

▶ **Definition 1.** *A* permutation protocol $\mathcal{P} = (n, \mathcal{H}, \Gamma, A)$ *is given by a number n of* object cards*, a deck of helping cards $\mathcal{H}$ with initial arrangement $\Gamma\colon \{n+1, \dots, n+|\mathcal{H}|\} \to \mathcal{H}$, and a sequence A of* actions *where each action can be either*

- $(\mathsf{privatePerm}, \Pi)$ *for $\Pi \subseteq S_{n+|\mathcal{H}|}$ implemented with active security and choice, and respecting $\{1, \dots, n\}$ (i.e. $\forall \pi \in \Pi\colon \pi(\{1, \dots, n\}) = \{1, \dots, n\}$), or*
- (check, p, o) *for a position p of a helping card (i.e. $n < p \leq n + |\mathcal{H}|$) and an expected outcome $o \in \mathcal{H}$.*

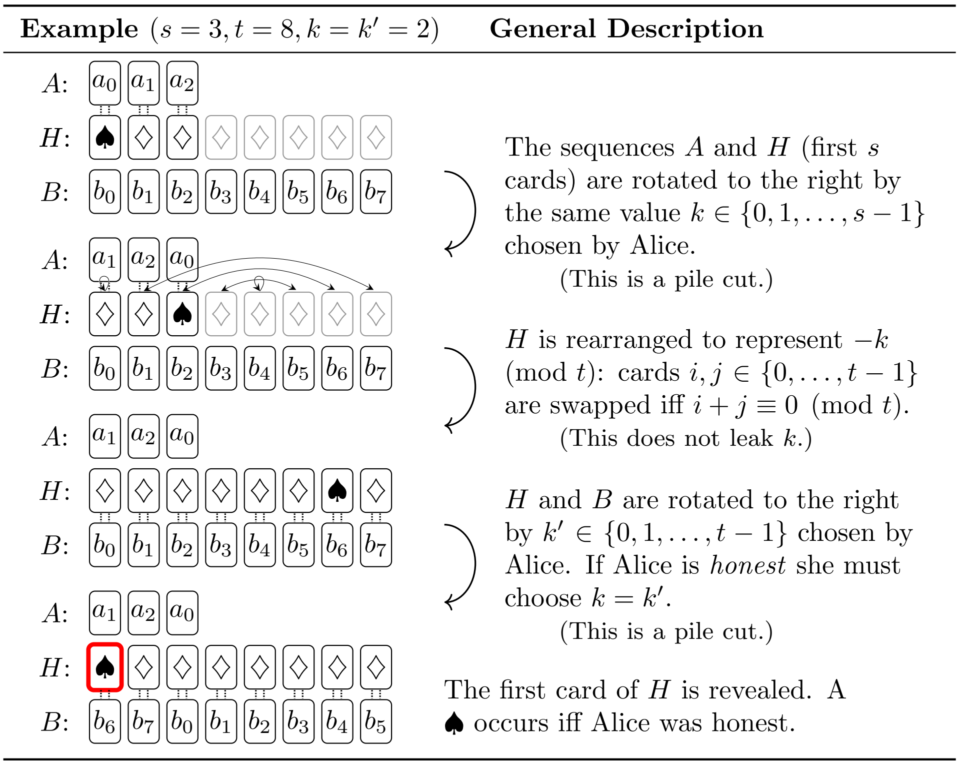
Indeed, consider the following procedure: We start with n object cards lying on a table (positions $1, \dots, n$). We place the sequence Γ next to it, at positions $n+1, \dots, n+|\mathcal{H}|$, and go through the actions of $\mathcal{P}$. Whenever the action $(\mathsf{privatePerm}, \Pi_i)$ is encountered, we use the procedure $\mathcal{P}_i$ implementing Π_i to let Alice apply a permutation on the current sequence. When an action (check, p, o) is encountered, the p-th card is revealed. If its symbol is o, Bob continues, otherwise he aborts, declaring Alice as dishonest. In the end, the helping cards are removed, yielding a permuted sequence of object cards. (All permutations respect $\{1, \dots, n\}$, hence, the helping and the object cards remain separated).

We are interested in the set $\mathsf{comp}(\mathcal{P}) \subseteq S_{n+|H|}$ of permutations *compatible* with $\mathcal{P}$. If there are k $\mathsf{privatePerm}$ actions with permutations sets $\Pi_1, \dots, \Pi_k$ and $\pi_i \in \Pi_i$, then $\pi_k \circ \dots \circ \pi_1$ is compatible with $\mathcal{P}$ if each check *succeeds*, meaning if (check, p, o) happens after the i-th $\mathsf{privatePerm}$ action (and before the $i+1$st, if $i < k$) then $\Gamma[(\pi_i \circ \dots \circ \pi_1)^{-1}(p)] = o$. We argue that this implements $\Pi' = \mathsf{comp}(\mathcal{P})|_{\{1,\dots,n\}}$ using $\mathcal{H}$ (and, possibly, helping cards to implement Π_i).

Alice can freely pick any $\pi' \in \Pi'$; using an appropriate decomposition, all checks will succeed. In this case, Bob knows that the performed permutation is from Π'. No player learns anything about the object cards (only helping cards are turned) and conditioned on Alice being honest, the outcome of the checks is determined, so Bob learns nothing about π'.

Coupled Rotations

Let $\varphi = (1\ 2\ \dots\ s)$, $\psi = (s{+}1\ s{+}2\ \dots\ s{+}t)$, and assume $s < t$. For $\pi = \psi \circ \varphi = \varphi \circ \psi$ we call $\Pi = \{\pi^k \mid 0 \leq k < s\}$ the *coupled rotation* with parameters s and t. Note that Π is not a group since $\pi^s \notin \Pi$. We aim to implement Π. We make use of a helping deck $[\![\spadesuit, (t{-}1) \cdot \heartsuit]\!]$ available in positions $H = \{h_0, h_1, \dots, h_{t-1}\}$ with ♠ at position h_0. Then define $\hat{\varphi} := \varphi \circ (h_0\ \dots\ h_{s-1})$ and $\hat{\psi} := \psi \circ (h_0\ \dots\ h_{t-1})^{-1}$ and consider the permutation protocol $\mathcal{P}$ in Figure 5 (left), and Figure 4 for illustration. The idea here is that Alice may choose k and k' and perform $\hat{\varphi}^k$ and $\hat{\psi}^{k'}$ to the sequence. However, k is "recorded" in the


A:
H:
B:
a_0 a_1 a_2
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
a_1 a_2 a_0
b_6 b_7 b_0 b_1 b_2 b_3 b_4 b_5

Example ($s = 3, t = 8, k = k' = 2$) **General Description**

The sequences A and H (first s cards) are rotated to the right by the same value $k \in \{0, 1, \dots, s-1\}$ chosen by Alice.
(This is a pile cut.)

H is rearranged to represent $-k \pmod t$: cards $i, j \in \{0, \dots, t-1\}$ are swapped iff $i + j \equiv 0 \pmod t$.
(This does not leak k.)

H and B are rotated to the right by $k' \in \{0, 1, \dots, t-1\}$ chosen by Alice. If Alice is *honest* she must choose $k = k'$.
(This is a pile cut.)

The first card of H is revealed. A ♠ occurs iff Alice was honest.

Figure 4 The sequence A of length s and B of length t are to be rotated by the same value k chosen privately by Alice. A helping sequence ensures that the same value is used. All cards are face-down, except for the highlighted card in the last step. The dotted lines indicate that cards are belonging to the same pile in a pile cut, i.e. they maintain their relative position during the cut. The rearrangement of the helping cards is useful in this visualization (so that H and B can be rotated in the same direction) but is not reflected in the formal description.

configuration of a helping sequence and $-k'$ is "added" on top. A check ensures that the helping sequence is in its original configuration, implying $k = k'$ as required. Note that $\langle\hat{\varphi}\rangle$ and $\langle\hat{\psi}\rangle$ are pile cuts, which we already know how to implement. In total, we implemented

$$\begin{aligned}\mathsf{comp}(\mathcal{P}) &= \{\hat{\psi}^{k'} \circ \hat{\varphi}^{k} : 0 \le k < s, 0 \le k' < t, \Gamma[(\hat{\psi}^{k'} \circ \hat{\varphi}^{k})^{-1}(h_0)] = \spadesuit\}|_{\{1,\dots,n\}}\\ &= \{\hat{\psi}^{k'} \circ \hat{\varphi}^{k} : 0 \le k < s, 0 \le k' < t, k' = k\}|_{\{1,\dots,n\}}\\ &= \{\psi^{k} \circ \varphi^{k} : 0 \le k < s\}|_{\{1,\dots,n\}} = \Pi.\end{aligned}$$

Products, Conjugates and Syntactic Sugar

The protocol in Figure 5 (middle) implements $\Pi_2 \circ \Pi_1$ using Π_1 and Π_2, showing that if Π_1 is implemented using $\mathcal{H}_1$ and Π_2 is implemented using $\mathcal{H}_2$, then $\Pi_2 \circ \Pi_1$ is implemented using $\mathcal{H}_1 \cup \mathcal{H}_2$. As a corollary, if Π is implemented using $\mathcal{H}$ then so is any conjugate $\Pi' = \{\pi^{-1}\} \circ \Pi \circ \{\pi\}$. Figure 5 (right) uses (perm, π) instead of $(\mathsf{privatePerm}, \{\pi\})$ to emphasize that such deterministic actions can be carried out publicly.

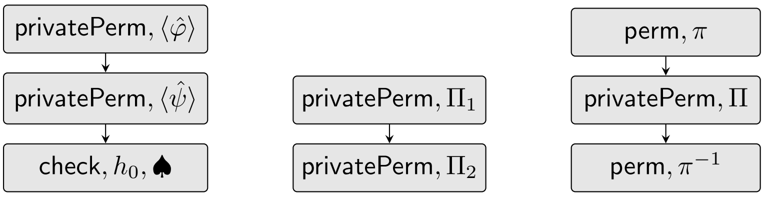


Figure 5 Protocols implementing a coupled rotation (*left*), the product of two permutation sets (*middle*) and the conjugation of a permutation set (*right*).

Generalized Coupled Rotations

We generalize the idea of a coupled rotation to more than two sequences. Let $\pi \in S_n$ with cycle decomposition $\pi = \varphi_0 \circ \cdots \circ \varphi_\ell$ for $\ell \geq 2$ and increasingly ordered cycle lengths $t_0 \leq t_1 \leq t_2 \leq \ldots \leq t_\ell$. We aim to implement $\Pi = \{\pi^k \mid 0 \leq k < t_0\}$ using $t_\ell + 2 \cdot t_0$ helping cards, originally available in the following positions which we label as shown.

$$\underbrace{\overset{\spadesuit}{m_0}\ \ \overset{\diamondsuit}{m_1}\ \ \overset{\cdots}{\ldots}\ \ \overset{\diamondsuit}{m_{t_\ell-1}}}_{\text{main}}\ \underbrace{\overset{\spadesuit}{x_0}\ \ \overset{\diamondsuit}{x_1}\ \ \overset{\cdots}{\ldots}\ \overset{\diamondsuit}{x_{t_0-1}}}_{\text{temp}}\ \underbrace{\overset{\spadesuit}{s_0}\ \ \overset{\diamondsuit}{s_1}\ \ \overset{\cdots}{\ldots}\ \overset{\diamondsuit}{s_{t_0-1}}}_{\text{store}}$$

We think of the three areas as "registers" *containing* values indicated by the position of ♠ (initially 0). The registers have associated rotations:

$$\psi_{\mathsf{temp}} := (x_0\ \ldots\ x_{t_0-1}), \qquad \psi_{\mathsf{store}} := (s_0\ \ldots\ s_{t_0-1}), \qquad \psi_i := (m_0\ \ldots\ m_{t_i-1}).$$

The protocol's idea is that Alice performs $\varphi_0^{k_0} \circ \cdots \circ \varphi_\ell^{k_\ell}$ and checks will ensure $k_0 = k_1 = \ldots = k_\ell$. To this end, k_0 is recorded in the store register (we use $\langle\varphi_0 \circ \psi_{\mathsf{store}}\rangle$). Then, for each round $i \in \{1, 2 \ldots, \ell - 1\}$ the value k_0 is cloned into the main register by first swapping it to the temp register and then moving it to the store *and* main register using $\psi_{\mathsf{copy}} := \psi_{\mathsf{temp}}^{-1} \circ \psi_{\mathsf{store}} \circ \psi_0$. The cloned copy of k_0 in main is consumed when forcing Alice to do $\widehat{\varphi}_i{}^{k_0}$ where $\widehat{\varphi}_i := \varphi_i \circ \psi_i^{-1}$. The last round is similar. Using the following two swappings, the protocol is formally given in Figure 6.

$$\pi_{\mathsf{store}\leftrightarrow\mathsf{tmp}} := (s_0\ x_0) \ldots (s_{t_0-1}\ x_{t_0-1}), \pi_{\mathsf{store}\leftrightarrow\mathsf{main}} := (s_0\ m_0) \ldots (s_{t_0-1}\ m_{t_0-1}).$$

We now check that this implements the generalized coupled rotation Π using the helping cards $[\![3 \cdot \spadesuit, (t_\ell{+}2t_0{-}3) \cdot \diamondsuit]\!]$, cf. Appendix A in the full version [10]. The main ingredient is the loop invariant:

If $\pi \in S_{n+2t_0+t_\ell}$ is compatible with the actions until after the i-th execution of the loop and S is the starting sequence then there exists $k \in \{0, \ldots, t_0 - 1\}$ such that:

- $\pi|_{\{1,\ldots,n\}} = \varphi_i^k \circ \ldots \circ \varphi_1^k \circ \varphi_0^k$,
- *in $\pi(S)$ all registers contain* 0 *except for store, which contains k.*

We remark that by introducing additional check steps, any subset of a generalized coupled rotation can be implemented as well.

Subgroups of S_n

Generalized coupled rotations are sufficient for:

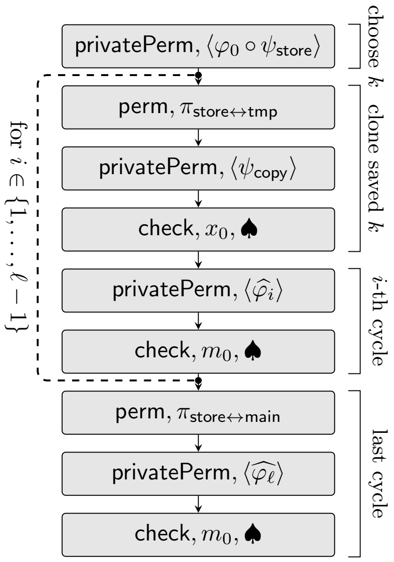

privatePerm, $\langle \varphi_0 \circ \psi_{\mathsf{store}} \rangle$
perm, $\pi_{\mathsf{store}\leftrightarrow\mathsf{tmp}}$
privatePerm, $\langle \psi_{\mathsf{copy}} \rangle$
check, x_0, ♠
privatePerm, $\langle \widehat{\varphi_i} \rangle$
check, m_0, ♠
perm, $\pi_{\mathsf{store}\leftrightarrow\mathsf{main}}$
privatePerm, $\langle \widehat{\varphi_\ell} \rangle$
check, m_0, ♠
choose k
clone saved k
i-th cycle
last cycle
for $i \in \{1, \ldots, \ell - 1\}$

Figure 6 Protocol to implement a generalized coupled rotation with $\ell + 1$ cycles of length $t_0, t_1, \ldots, t_\ell$. Notation is explained in the text.

▶ **Proposition 2.** *Any subgroup Π of S_n can be implemented with active security and choice using only the helping deck $[\![3 \cdot ♠, (n-3) \cdot ♢]\!]$ for (generalized) coupled rotations and the helping deck $[\![♠, (n-1) \cdot ♢]\!]$ for (pile) cuts.*

Proof. Note that $\Pi = \prod_{\pi \in \Pi} \langle \pi \rangle$, i.e. Π can be written as the product of cyclic subgroups. Moreover, any cyclic subgroup can be written as $\langle \pi \rangle = \{\pi^0, \ldots, \pi^{k-1}\}^\ell$, where k is the length of the shortest cycle in the cycle decomposition of π and $\ell = \lceil \mathsf{ord}(\pi)/(k-1) \rceil$. Hence, Π can be written as the product of rotations and (generalized) coupled rotations, each of which are implemented with the required helping decks. Using the implementation of products (page 9), we are done. ◀

A *simple* decomposition of Π into products of previously implemented permutation sets is desirable to keep the permutation protocol simple. We do not consider this here and merely state that $|\Pi|$ is an upper bound on the number of terms.

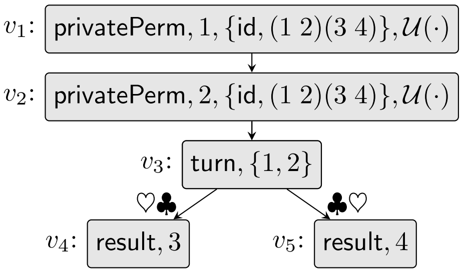
5 Computational Model with Two Players

In the following, two players jointly manipulate a sequence of cards to compute a (possibly randomized) function, i.e. they transform an input sequence into an output sequence. Both have incomplete information about the execution and the goal is to compute with no player learning anything about input or output[5].

Two Player Protocols

A *two player protocol* is a tuple $(\mathcal{D}, U, Q, A)$ where $\mathcal{D}$ is a deck, U is a set of input sequences, Q is a (possibly infinite, computable) rooted tree with labels on some edges, and $A\colon V(Q) \to$ Action is an action function that assigns to each vertex an action which can be perm, turn,

[5] An explanation of our security notions follows in Section 6.


v_1: privatePerm, 1, {id, (1 2)(3 4)}, $\mathcal{U}(\cdot)$
v_2: privatePerm, 2, {id, (1 2)(3 4)}, $\mathcal{U}(\cdot)$
v_3: turn, {1, 2}
$\heartsuit\clubsuit$
$\clubsuit\heartsuit$
v_4: result, 3
v_5: result, 4

$$\mathcal{D} = \llbracket 3 \cdot \clubsuit, 2 \cdot \heartsuit \rrbracket,$$

$$\begin{aligned} U = \{&(\clubsuit, \heartsuit, \clubsuit, \clubsuit), \\ &(\clubsuit, \heartsuit, \heartsuit, \clubsuit), \\ &(\heartsuit, \clubsuit, \clubsuit, \clubsuit), \\ &(\heartsuit, \clubsuit, \heartsuit, \clubsuit)\}, \end{aligned}$$

Q, A: as shown on the left

Figure 7 A protocol example in the two player model, with possible execution trace: $(I = (\heartsuit, \clubsuit, \heartsuit, \clubsuit), O = (\heartsuit), \mathcal{T}_1 = (\mathsf{id}), \mathcal{T}_2 = ((1\ 2)(3\ 4)), W = (v_1, v_2, v_3, v_5))$. This is an actively secure implementation of the AND protocol in [14, Sect. 3.2]. The first two cards encode an input a as $(\clubsuit, \heartsuit) \triangleq 0$, $(\heartsuit, \clubsuit) \triangleq 1$, the third card encodes an input b as $\clubsuit \triangleq 0$, $\heartsuit \triangleq 1$. This encoding is also used for output $a \wedge b$.

result, and privatePerm, with parameters as explained below. All input sequences have the same length n and are formed by cards from $\mathcal{D}$. Vertices with a perm or privatePerm action have exactly one child, vertices with a result action have no children, and those with a turn action have one child for each possible sequence of symbols the turned cards might conceal, and the edge to that child is annotated with that sequence.

When a protocol is *executed on an input sequence* $I \in U$, we start with the face-down sequence $\Gamma = I$ at the root of Q and empty *permutation traces* $\mathcal{T}_1$ and $\mathcal{T}_2$ for players 1 and 2, respectively. Execution proceeds along a descending path in Q and for each vertex v that is encountered, the action $A(v)$ is executed on the current sequence of cards:

(perm, π) for a permutation $\pi \in S_n$. This replaces the current sequence Γ by the permuted sequence $\pi(\Gamma)$. Execution proceeds at the unique child of v.

(turn, T) for some set $T \subseteq \{1, \dots, n\}$. For $T = \{t_1 < t_2 < \dots < t_k\}$, the cards $\Gamma[t_1], \dots, \Gamma[t_k]$ are turned face-up, revealing their symbols. The vertex v must have an outgoing edge labeled $(\Gamma[t_1], \dots, \Gamma[t_k])$. Execution proceeds at the corresponding child after the cards are all turned face-down again.

(privatePerm, $p, \Pi, \mathcal{F}(\cdot)$) for a player $p \in \{1, 2\}$, a permutation set $\Pi \subseteq S_n$ and $\mathcal{F}$ being a parameterized distribution on Π. Formally, $\mathcal{F}$ is a function that maps the current permutation trace $\mathcal{T}_p$ of player p to a distribution $\mathcal{F}(\mathcal{T}_p)$ on Π. If $\mathcal{F}(\mathcal{T}_p)$ is the uniform distribution on Π for each $\mathcal{T}_p$ we denote this as $\mathcal{U}(\cdot)$. Player p picks a permutation $\pi \in \Pi$. The current sequence Γ is replaced by the permuted sequence $\pi(\Gamma)$ and π is appended to the player's permutation trace $\mathcal{T}_p$. If player p is *honest* she picks π according to $\mathcal{F}(\mathcal{T}_p)$. Execution proceeds at the unique child of v.

(result, $p_1, \dots, p_k$) for distinct positions $p_1, \dots, p_k \in \{1, \dots, n\}$. Execution terminates with *output* $O = (\Gamma[p_1], \dots, \Gamma[p_k])$ encoded by face-down cards.

The execution yields an *execution trace* $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$, containing input, output, permutation traces of the players and the descending path W in Q that was taken, cf. Figure 7. The output of non-terminating protocols is $O = \bot$. Note that we will use permutation protocols from Section 4 in the privatePerm steps, however we use them as black boxes. In particular, the actions specific to permutation protocols (e.g. check) are not part of two player protocols. We say $\mathcal{P}$ is *implemented* using a helping deck $\mathcal{H}$ if each permutation set of a privatePerm action is implemented using $\mathcal{H}$ (as in Section 3). The way we define it, existence, implementability and security of a protocol are separate issues. Security is discussed next.

6 Passive and Active Security

Intuitively, an implemented protocol is (information-theoretically) secure if no player can derive any statistical information about input or output from the choices and observations they make during the execution of the protocol. So the first question is, what information does a player obtain, say Alice, that could potentially be relevant? At first we consider the setting where both players are honest. Surely, Alice knows the public information W, i.e. the execution path of the protocol run, in which the sequence of actions and their parameters are implicit. For each action along W she may have obtained additional information during its execution. To get a complete picture, we go through all types of actions:

- turn actions reveal some card symbols. However, as each outcome corresponds to a unique child vertex where execution continues, this information is already implicit in W.
- perm actions are deterministic and reveal no information. The same is true for result actions. Note that they only *indicate* the position of the output, not reveal it.
- For privatePerm actions, the observations that can be made depend on the implementation. If the protocols are implemented in our sense (see Section 3) and Alice is the active player then Alice learns nothing of relevance except her own choice of permutation (which is recorded in her permutation trace) and, since Alice is honest, Bob learns nothing at all.

So the only potentially relevant information player p has with regards to input and output is W and $\mathcal{T}_p$. Therefore it is adequate to define:

▶ **Definition 3** (Passive Security). *A two player protocol $\mathcal{P} = (\mathcal{D}, U, Q, A)$ is* secure against passive attackers *if for any random variable $I \in U$ the following holds: If $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$ is the execution trace when executing $\mathcal{P}$ with honest players on input I, then (I, O) is independent of $(\mathcal{T}_p, W)$ for both $p \in \{1, 2\}$.*

Delegated Computation

Passive security implies that if a player has no prior knowledge about in- or output, executing the protocol leaves her in this oblivious state. In particular, by following the protocol the players implement what we call an *oblivious delegated computation* where the computation is performed on secret data (provided by a third party), and the output is not revealed to the executers.

Note that this setting differs from the *standard multiparty computation setting*, where players provide part of the input and usually the output is sent to the players in non-committed (non-hiding) form, i.e., learned by the players. In this case, security means that the players learn nothing *except* what can be deduced from the facts they are permitted to know. It is important to understand that our definition is still adequate for such cases, as *any protocol that is secure in the delegated computation setting is also secure if players have (partial) information about input and output.* The formal reason is the basic fact that for any event E relating only to (I, O), i.e., E is independent of $(\mathcal{T}_p, W)$, conditioning the probability space on E will retain the independence of (I, O) and $(\mathcal{T}_p, W)$.

Moreover, protocols secure in the delegated setting are flexibly applicable in different contexts, making it a very suitable framework. For example, non-delegateable (non-committed input format) protocols which can only be performed by players knowing the input (cf. [13, 23, 22]) cannot be transferred to the delegated setting and are hence unsuitable for use with hidden intermediate results from previous computations. Hence, we protect the output and do not assume knowledge of the inputs. This is a natural setting for card-based cryptography, as all committed-format protocols in the literature achieve this notion, it ensures that the protocols can be used in larger protocols, and it is at least as secure as the other notions, due to the information-theoretic setting.

The above definition of passive security is sufficient if players can be trusted to properly execute the protocol. In that case any privatePerm action can directly be performed by the specified player while the other player looks away. Of course, our main concern here is the situation where looking away is not an option.

Permutation Security and Active Security

To argue about security in the presence of a malicious player, we must first discuss what such a player may do. Doing this rigorously would require to closely model the physical world, which allows for different threats than in the usual cryptographic settings. We certainly have to assume physical restrictions, as otherwise we cannot achieve anything.[6] For example, as our security relies on the possibility of keeping face-down cards, we must assume that an attacker does not resort to certain radical means that immediately and unambiguously identify her as an attacker. (However, note that we can protect against such active attackers which turn over cards by the generic "private circuit" compiler due to Ishai, Sahai, and Wagner [5].) Hence, we can assume that she does not interfere with the correct execution of perm and turn actions, nor does she, in open violation of the protocol, spontaneously seize or turn over some of the cards or mark them in any way.

On the other hand we can plausibly argue that certain mechanisms are sufficient to counter attacks other than those that our paper is concerned with. We may argue that the cards could be put into envelopes, and any attempt to reveal its contents contrary to the protocol will be countered by the cautious other players jumping in to physically abort the protocol in that case.

Concerning an operation $(\mathsf{privatePerm}, \text{Alice}, \Pi, \mathcal{F}(\cdot))$ with implemented Π, there is by definition of *implemented permutation set* no possibility for Alice to perform a permutation $\pi \notin \Pi$. If she causes a permutation protocol to fail, Bob aborts the execution before any sensitive information is revealed. Otherwise, Alice is limited to disrespecting $\mathcal{F}(\cdot)$. This is captured as follows:

▶ **Definition 4.** *Let $\mathcal{P} = (\mathcal{D}, U, Q, A)$ be a two player protocol.*

(i) *A* permutation attack ξ on $\mathcal{P}$ as player $p \in \{1, 2\}$ *specifies for each vertex $v \in V(Q)$ with an action of the form $A(v) = (\mathsf{privatePerm}, p, \Pi, \mathcal{F}(\cdot))$, a permutation $\xi(v) \in \Pi$. Replacing such $\mathcal{F}(\cdot)$ with the (point) distributions that always choose $\xi(v)$, yields the* attacked protocol $\mathcal{P}^\xi$.

(ii) *An attack ξ is* unsuccessful *if the following holds. Whenever $I \in U$ is a random variable denoting an input and $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$ and $(I, O^\xi, \mathcal{T}_1^\xi, \mathcal{T}_2^\xi, W^\xi)$ are the resulting execution traces of $\mathcal{P}$ and $\mathcal{P}^\xi$, then for any values i, o, w:*

$$\Pr[W^\xi{=}w] > 0 \Longrightarrow \Pr[(I, O^\xi){=}(i, o) \mid W^\xi{=}w] = \Pr[(I, O){=}(i, o)]. \qquad (\star)$$

(iii) *We say $\mathcal{P}$ is* secure against permutation attacks *if each permutation attack on $\mathcal{P}$ is unsuccessful.*

In light of our discussion above we finally define:

▶ **Definition 5.** *A two player protocol $\mathcal{P} = (\mathcal{D}, U, Q, A)$ has an* actively secure implementation *if each permutation set Π occurring in a* privatePerm *action is implemented and $\mathcal{P}$ is secure against permutation attacks.*

[6] We do not get ultimately strong guarantees for the physical actions such as in quantum cryptography, where, if (a subset of) quantum theory is true, no adversary can predict a randomness source, no matter what she does physically.

Intuitively, a protocol has permutation security if: No matter what permutations a player chooses ($\forall \xi$), and no matter what the turn actions end up revealing ($\forall W^\xi$), the best guess for the in- and output (distribution of (I, O^ξ), given W^ξ) is no different from what he would have said, had he not been involved in the computation at all (distribution of (I, O)). We make a few remarks.

- Passively secure protocols terminate almost surely, otherwise $O = \bot$ can be recognized from an infinite path W. For similar reasons, a permutation attacker can never cause a protocol with permutation security to run forever.[7]
- In our definition, permutation attackers are deterministic without loss of generality. Intuitively, if an attacker learns nothing *no matter what* ξ *she chooses*, then choosing ξ *randomly* is just a fancy way of determining in what way she is going to learn nothing.
- For similar reasons, permutation security implies passive security, since playing honestly is just a weighted mixture of "pure" permutation attacks.
- We cannot say anything if *both* players are dishonest or if they share their execution traces with one another. We also cannot guarantee that player learns nothing if the *other* player is dishonest.

Permutation Security from Passive Security

There is an important special case in which the powers of a permutation attacker turn out to be ineffective, namely if the distributions $\mathcal{F}(\mathcal{T}_p)$ never assign zero probability to a permutation.

▶ **Proposition 6.** *Let $\mathcal{P} = (\mathcal{D}, U, Q, A)$ be a passively secure two player protocol where for each action of form $(\mathsf{privatePerm}, p, \Pi, \mathcal{F}(\cdot))$ and each permutation trace $\mathcal{T}_p$ of player p, $\mathcal{F}(\mathcal{T}_p)$ has support Π*[8]*. If for each attack ξ the attacked protocol $\mathcal{P}^\xi$ terminates with probability 1*[9]*, then $\mathcal{P}$ is secure against permutation attacks.*

Proof. Consider an attack ξ on $\mathcal{P}$ as player $p \in \{1, 2\}$, let $I \in U$ be any random variable denoting an input and $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$ and $(I, O^\xi, \mathcal{T}_1^\xi, \mathcal{T}_2^\xi, W^\xi)$ be the execution traces of $\mathcal{P}$ and $\mathcal{P}^\xi$. Let w be any path in Q with $\Pr[W^\xi = w] > 0$ and t the permutation trace that ξ prescribes for player p along w (whenever $W^\xi = w$, then $\mathcal{T}_p^\xi = t$). For any i, o we have:

$$\begin{aligned}\Pr[(I, O^\xi) = (i, o) \mid W^\xi = w] &= \Pr[(I, O^\xi) = (i, o) \mid (\mathcal{T}_p^\xi, W^\xi) = (t, w)] \\ &= \Pr[(I, O) = (i, o) \mid (\mathcal{T}_p, W) = (t, w)] \\ &= \Pr[(I, O) = (i, o)].\end{aligned}$$

From the first to the second line, note that firstly, since w is finite, the sequence t of choices is finite as well, so, using the assumption that $\mathsf{supp}(\mathcal{F}(\mathcal{T}_p)) = \Pi$ in all cases, there is some positive probability that an honest player behaves exactly like the attacker with respect to this finite sequence of choices. Therefore, the conditional probability in the second line is well defined. Secondly, the attacked protocol and the original protocol behave alike once we fix the behavior of player p so we have the stated equality. From the second to the third line we use the passive security of $\mathcal{P}$. ◀

[7] Protocols that almost surely output $\bot$ are a pathological exception.
[8] Otherwise, active attackers may pick $\pi \in \Pi$ which honest players never choose.
[9] this excludes a pathological case

7 Implementing Mizuki–Shizuya Protocols

In [17], Mizuki and Shizuya's self-proclaimed goal was to define a "computational model which captures what can possibly be done with playing cards". Hence, any secure real-world procedure to compute something with playing cards can be formalized as a secure protocol in their model.[10] The other direction is not so clear. Given a secure protocol in the model, can it be implemented in the real world? We believe the answer is probably "no" (or, at least, not clearly "yes"). However, our work of identifying implementable actions in the two player model implies that a very natural subset of actions in Mizuki and Shizuya's model is implementable, even with active security: *uniform closed shuffles* (see below). Note that these shuffles already allow for securely computing any circuit [19].

Mizuki–Shizuya Protocols

We modify Mizuki and Shizuya's model slightly: instead of state machine semantics we stick to a tree of actions as in the two player model. This is an equivalent way of defining protocols, cf. [7, Sects. 3 and 4].

A *Mizuki–Shizuya protocol* is a tuple $\mathcal{P} = (\mathcal{D}, U, Q, A)$ similar to a two player protocol. The actions perm, result and turn are available as before, but instead of privatePerm actions there are shuffle actions of the form $(\mathsf{shuffle}, \Pi, \mathcal{F})$ where Π is a set of permutations and $\mathcal{F}$ is a probability distribution on Π. Executing a protocol works as before, but there are no separate permutation traces for players (there are no players at all), instead there is a single permutation trace $\mathcal{T}$. The actions perm, turn and result work as before. When an operation $(\mathsf{shuffle}, \Pi, \mathcal{F})$ is encountered, a permutation $\pi \in \Pi$ is chosen according to $\mathcal{F}$ (independent from previous choices). This permutation π is applied to the current sequence of cards without anyone learning π and appended to the permutation trace $\mathcal{T}$.

For any input $I \in U$, an execution of a protocol is described by the execution trace $(I, O, \mathcal{T}, W)$ where O is the output ($O = \perp$ if it did not terminate), $\mathcal{T}$ the permutation trace and W the path of the execution in Q. It is assumed that only W is observed, suggesting the following security notion:

▶ **Definition 7** (Security of Mizuki–Shizuya Protocols). *A Mizuki–Shizuya protocol $\mathcal{P}$ is* secure *if for each random variable $I \in U$ and resulting execution trace $(I, O, \mathcal{T}, W)$ of the protocol, (I, O) is independent from W.*

Implementing Uniform Closed Mizuki–Shizuya Protocols

We call a shuffle $(\mathsf{shuffe}, \Pi, \mathcal{F})$ *uniform* if $\mathcal{F}$ is the uniform distribution on Π, and *closed* if Π is a group. We call a Mizuki–Shizuya protocol *uniform closed* if each of its shuffle actions is uniform and closed. We are ready to state our main theorem.

▶ **Main Theorem.** *Let $\mathcal{P} = (\mathcal{D}, U, Q, A)$ be a secure uniform closed Mizuki–Shizuya protocol. Then there is a two player protocol $\hat{\mathcal{P}} = (\mathcal{D}, U, \hat{Q}, \hat{A})$ with actively secure implementation computing the same (possibly randomized) function as $\mathcal{P}$.*

Moreover, the implementation of $\hat{\mathcal{P}}$ uses as helping deck only $[\![3 \cdot \spadesuit, (n-3) \cdot \heartsuit]\!]$ for (generalized) coupled rotations and $[\![\spadesuit, (n-1) \cdot \heartsuit]\!]$ for chosen (pile) cuts. Here, n is the length of the input sequences.

[10] Excluding the use case of non-committed input protocols from [13] and [23], where the input is provided by a choice of privatePerm operations by a player, requiring input awareness/knowledge.

We sketch the proof here and give the formal proof in Appendix B in the full version [10]. Each uniform closed shuffle $(\mathsf{shuffle}, \Pi, \mathcal{U})$ of $\mathcal{P}$ is replaced by two actions $(\mathsf{privatePerm}, p, \Pi, \mathcal{U})$ for $p \in \{1, 2\}$. For $\pi_2 \circ \pi_1$ to be uniformly random in Π, it suffices if π_1 *or* π_2 is chosen uniformly random in Π (while the other is known). Therefore, the joint permutation applied to the sequence after both privatePerm actions looks uniformly random to both players. Hence, they learn nothing from the execution of $\hat{\mathcal{P}}$ that they would not have also learned from executing $\mathcal{P}$. Since $\mathcal{P}$ is secure, $\hat{\mathcal{P}}$ is passively secure and by Proposition 6 also secure against permutation attacks. Moreover, by Proposition 2 all Π are implemented using the stated helping decks, so $\hat{\mathcal{P}}$ has an actively secure implementation.

8 Active Input Security

In Section 6 we have argued that results for the delegated computation setting are also applicable when players have (partial) knowledge of the input and we narrowed our focus accordingly. In this section we take a second look at protocols where players provide the input themselves. In some cases, this allows for simpler protocols with fewer cards, but it also brings about specific issues regarding active security.

We do *not* attempt a formal definition of active security in this setting, leaving this open for future work. To simplify notation, we restrict the presentation to cases with two possible inputs for each player, denoted by 0 and 1.

Warm Up: Inputs in Standard Encoding

The standard encoding of binary inputs uses the card sequence ♣♡ to represent 0 and ♡♣ to represent 1. When expected to provide an input in this format, a malicious player could provide marked cards or cards with altogether different symbols. Mizuki and Shizuya [18] give special attention to detecting the inputs ♣♣ and ♡♡ that a malicious player might provide when given several copies of otherwise uncompromised cards.

A simple solution is to place, for each player, the sequence ♣♡ on the table and give the player the opportunity to swap the cards with no other player noticing. This is a chosen cut and has an actively secure implementation as discussed in Section 3.1, though, arguably, simpler procedures exist for this special case. After all players have provided their inputs, an ordinary protocol expecting inputs in standard format is started.

Input by Permutation

We now turn to protocols that request the inputs of the players sequentially, and by performing a permutation on the cards. In one case, we even require players to provide their input more than once.

We capture this with an additional formal action of the form $(\mathsf{inputPerm}, p, \pi_0, \pi_1)$. When it is encountered, player p should permute the current card sequence using the permutation π_0, if his input is 0, and using π_1, if his input is 1. His choice should stay hidden from the other players. Note that [9, Sect. 12.9] specifies a more general form of this action (not limited to inputs of one bit), as well as a more general form of the corresponding state diagrams (see below). Here, we chose to use a simplified version for ease of exposition.

Considered in isolation, the action is essentially identical to $(\mathsf{privatePerm}, p, \{\pi_0, \pi_1\})$, however, its role in the surrounding protocol and its relation to security notions is fundamentally different: In the case of inputPerm, the player's choice corresponds to her input and may affect the output, while in the case of privatePerm, the choice is (in a secure protocol) independent of input and output and may be (indirectly) leaked in subsequent actions.

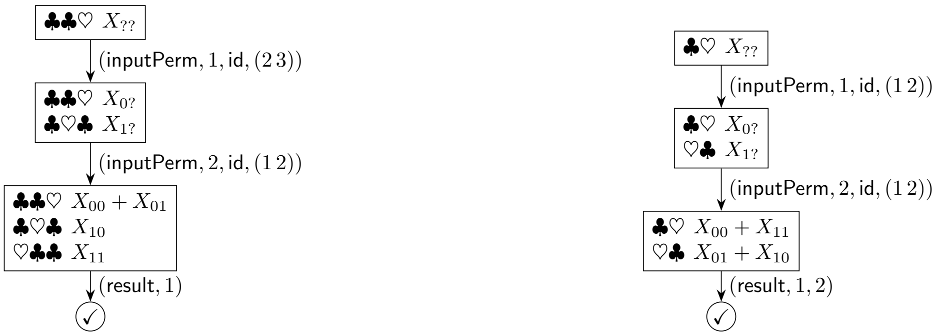


Figure 8 On the *left*, we show the state diagram of a three-card AND protocol [13, Sect. 3.2] with inputs provided by players. The result operation indicates that the first card is the output. Here, ♡ stands for 1 and ♣ for 0. On the *right* is the state diagram of a two-card XOR protocol of [22] in a similar spirit but the output is in standard format.

During a protocol execution, let the *input trace* $\mathcal{I}_p$ of a player p be the sequence of permutations performed by p during her inputPerm actions encountered so far. To simplify some diagrams in the following, we write ? for the empty input trace and 0 or 1 for non-empty input traces of players that have (so far) always chosen the permutation corresponding to the same input, i.e., have always chosen π_0 or always π_1.

Two Simple Examples: AND and XOR

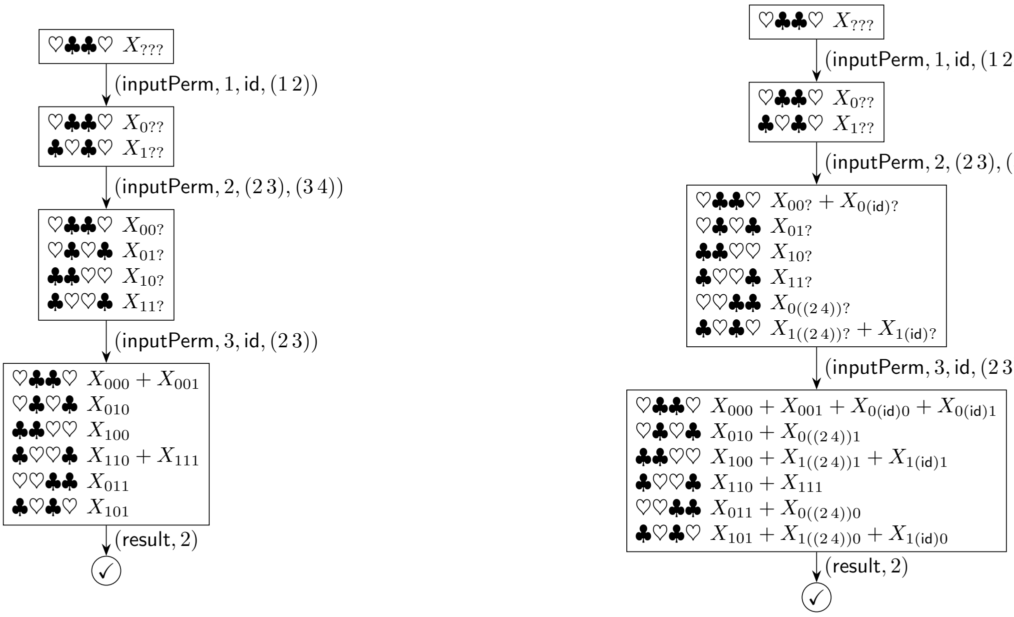
We consider two very simple protocols for computing AND and XOR from [13, Sect. 3.2] and [22], respectively, shown in Figure 8. The AND protocol starts with the sequence ♣♣♡. The first player is expected to perform id or (2 3) if his input is 0 or 1, respectively. The second player acts similarly, but on the first two cards. In total, the ♡ is moved to the first position if and only if both players choose the permutation corresponding to input 1. The card in the first position therefore encodes the AND of the two player's input bits. Active security can be achieved since inputPerm actions correspond to chosen cuts. The XOR protocol is even simpler and easily generalizes to more than two players.

The shown state diagrams are adapted from [12]. Roughly speaking, each state shows which combination of input traces give rise to which card sequence. For the purposes of this section, an intuitive understanding is sufficient. For instance, in the initial state of the AND protocol in Figure 8, the card sequence is ♣♣♡ and the input traces of both players are empty, i.e. they are $() = ?$. This is represented by $X_{??}$. In the second state the input trace of player 1 could be $0 = (\mathsf{id})$ or $1 = ((2\,3))$ while the second player's input trace is still empty. The two possibilities are represented by $X_{0?}$ and $X_{1?}$ and are given next to the corresponding possibilities for the card sequence. In the last state, we see that two possibilities for the input traces may lead to the same card sequence. That card sequence is correspondingly annotated with the sum of the two possibilities.

Majority Protocols and Two Types of Attacks

The majority function with an odd number of bits as input computes the value (0 or 1) that makes up at least half of the inputs. Recently, Nakai et al. [22] proposed a protocol for computing the majority of three bits using four cards with non-standard input and output format. For comparison, note that among protocols where inputs and outputs are given in standard format, the known protocol using the fewest cards for three-input majority is [26] with eight cards. In our terminology the protocol is given as the left state diagram of Figure 9. The authors plausibly claim security in the honest-but-curious setting. It is, however, unclear how active security could be achieved due to the permutation set $\{\pi_0 = (2\,3), \pi_1 = (3\,4)\}$ in the $\mathsf{inputPerm}$ action of player 2. We cannot think of a simple mechanism that allows the player to perform π_0 and π_1 but prevents him from doing id or $(2\ 4)$, and possibly also $(2\ 3\ 4)$ and $(2\ 4\ 3)$.[11]

On the right of Figure 9, we depict the state diagram in the case where player 2 can (illegally) perform id or $(2\ 4)$ without being detected. In this attack scenario, player 2 can, e.g., force the result to be 0 via applying id, when both other player's inputs are 1.



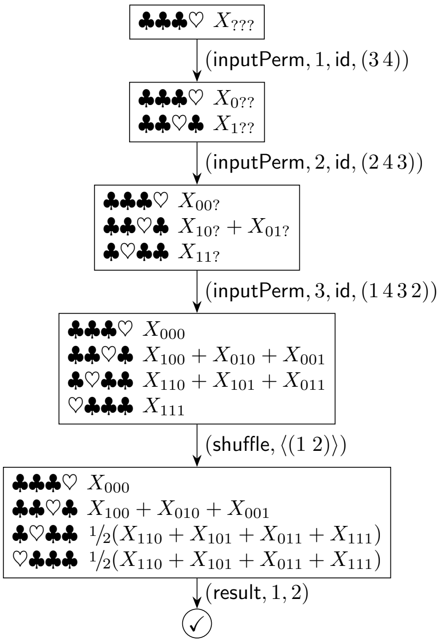
♡♣♣♡ $X_{???}$
$(\mathsf{inputPerm}, 1, \mathsf{id}, (1\,2))$
♡♣♣♡ $X_{0??}$
♣♡♣♡ $X_{1??}$
$(\mathsf{inputPerm}, 2, (2\,3), (3\,4))$
♡♣♣♡ $X_{00?}$
♡♣♡♣ $X_{01?}$
♣♣♡♡ $X_{10?}$
♣♡♡♣ $X_{11?}$
$(\mathsf{inputPerm}, 3, \mathsf{id}, (2\,3))$
♡♣♣♡ $X_{000} + X_{001}$
♡♣♡♣ X_{010}
♣♣♡♡ X_{100}
♣♡♡♣ $X_{110} + X_{111}$
♡♡♣♣ X_{011}
♣♡♣♡ X_{101}
$(\mathsf{result}, 2)$
✓
♡♣♣♡ $X_{???}$
$(\mathsf{inputPerm}, 1, \mathsf{id}, (1\,2))$
♡♣♣♡ $X_{0??}$
♣♡♣♡ $X_{1??}$
$(\mathsf{inputPerm}, 2, (2\,3), (3\,4))$
♡♣♣♡ $X_{00?} + X_{0(\mathsf{id})?}$
♡♣♡♣ $X_{01?}$
♣♣♡♡ $X_{10?}$
♣♡♡♣ $X_{11?}$
♡♡♣♣ $X_{0((2\,4))?}$
♣♡♣♡ $X_{1((2\,4))?} + X_{1(\mathsf{id})?}$
$(\mathsf{inputPerm}, 3, \mathsf{id}, (2\,3))$
♡♣♣♡ $X_{000} + X_{001} + X_{0(\mathsf{id})0} + X_{0(\mathsf{id})1}$
♡♣♡♣ $X_{010} + X_{0((2\,4))1}$
♣♣♡♡ $X_{100} + X_{1((2\,4))1} + X_{1(\mathsf{id})1}$
♣♡♡♣ $X_{110} + X_{111}$
♡♡♣♣ $X_{011} + X_{0((2\,4))0}$
♣♡♣♡ $X_{101} + X_{1((2\,4))0} + X_{1(\mathsf{id})0}$
$(\mathsf{result}, 2)$
✓

Figure 9 State diagram of the three-inputs majority protocol from [22] on the *left*. The second card encodes the result with ♡ standing for 1 and ♣ for 0. On the *right* we track the same protocol when player 2 is an active attacker who can illegally perform id or $(2\,4)$ during his $\mathsf{inputPerm}$ action.

In Figure 10 we give an alternative four-card majority protocol, which is conceptually very simple – similar to the AND protocol we saw before. Here each player cyclically rotates the so-far relevant cards by one for input 1 and does nothing otherwise. If the majority of the players did input 1, then the ♡ is in the first two positions. A shuffle of these two cards then conceals which one it was. The protocol has the advantage of only using $\mathsf{inputPerm}$ operations that are simple to implement with active security. The output is, however, encoded in an unusual way with ♣♣ representing 0 and ♡♣ and ♣♡ both representing 1. Note that there is a straightforward generalisation of the protocol to more than three inputs.

Finally, consider the three-card three-input majority protocol from [34] in Figure 11. All permutation sets $\{\pi_0, \pi_1\}$ of $\mathsf{inputPerm}$ actions can arguably be implemented with active security. However, since players 1 and 2 each have *two* $\mathsf{inputPerm}$ actions assigned to them, these players could choose their permutations incoherently, for instance, π_0 in the first $\mathsf{inputPerm}$ action and π_1 in the second. In the state diagram we have tracked this possible

[11] We can implement any $\mathsf{inputPerm}$ action if the two permutations are encoded as in [11], and a sort protocol is used to apply a chosen one of the two to the sequence of cards. For the present case this would, however, require at least 6 helping cards.



♣♣♣♡ $X_{???}$
(inputPerm, 1, id, (3 4))
♣♣♣♡ $X_{0??}$
♣♣♡♣ $X_{1??}$
(inputPerm, 2, id, (2 4 3))
♣♣♣♡ $X_{00?}$
♣♣♡♣ $X_{10?} + X_{01?}$
♣♡♣♣ $X_{11?}$
(inputPerm, 3, id, (1 4 3 2))
♣♣♣♡ X_{000}
♣♣♡♣ $X_{100} + X_{010} + X_{001}$
♣♡♣♣ $X_{110} + X_{101} + X_{011}$
♡♣♣♣ X_{111}
(shuffle, ⟨(1 2)⟩)
♣♣♣♡ X_{000}
♣♣♡♣ $X_{100} + X_{010} + X_{001}$
♣♡♣♣ $1/2(X_{110} + X_{101} + X_{011} + X_{111})$
♡♣♣♣ $1/2(X_{110} + X_{101} + X_{011} + X_{111})$
(result, 1, 2)
✓

Figure 10 State diagram of a three-inputs majority protocol. The idea is that players rotate the sequence by one, if their input is 1, or do nothing otherwise. In the end, if more than one player rotated, a heart ends up in the first or second position. An additional shuffle obscures which of both is the case. The first two cards encode the output with ♡♣ and ♣♡ standing for 1, while ♣♣ stands for 0.

attack for player 1, assuming the other players are honest. The occurrence of $X_{(\mathsf{id},\mathsf{id})00}$ at the outcome ♡♣♣ indicates that an output of 0 can occur even though players 2 and 3 have both input 0 if player 1 (illegally) chooses the identity permutation in both his inputPerm actions. It seems unlikely that there is a meaningful defense against such an attack that does not substantially alter the protocol.

The case of 3-bit majority protocols shows that the question of how many cards are required does not have a straightforward answer as it depends on the desired input and output formats, the security requirements, and, if active security is desired also on whether or not helping cards are counted that might be used in the *implementation* of the inputPerm operations.

9 Conclusion

Central to our notion of active security is the concept of a *permutation set implemented with active security and choice*, indicating that a player Alice can choose to perform a permutation from the set while Bob can know that Alice did not cheat, but nothing else. We argued that *cuts* and *pile cuts* have such an implementation and we used *permutation protocols* to build more sophisticated procedures handling any group of permutations. Moreover, we defined security for Mizuki–Shizuya protocols, active and passive security for our own *two player protocols* and showed how secure Mizuki–Shizuya protocols using only uniform closed shuffles can be transformed into actively secure two player protocols. This is a solid foundation for actively secure card-based cryptography.

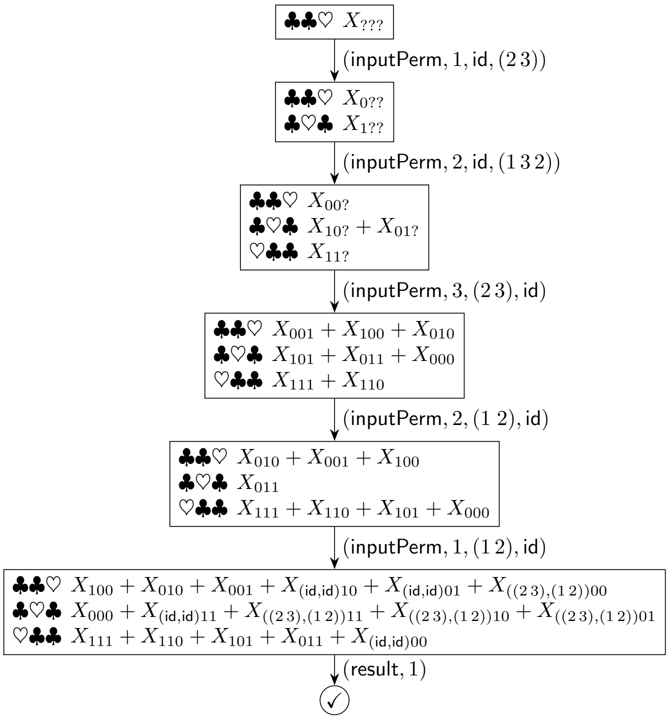

♣♣♡ $X_{???}$
(inputPerm, 1, id, (2 3))
♣♣♡ $X_{0??}$
♣♡♣ $X_{1??}$
(inputPerm, 2, id, (1 3 2))
♣♣♡ $X_{00?}$
♣♡♣ $X_{10?} + X_{01?}$
♡♣♣ $X_{11?}$
(inputPerm, 3, (2 3), id)
♣♣♡ $X_{001} + X_{100} + X_{010}$
♣♡♣ $X_{101} + X_{011} + X_{000}$
♡♣♣ $X_{111} + X_{110}$
(inputPerm, 2, (1 2), id)
♣♣♡ $X_{010} + X_{001} + X_{100}$
♣♡♣ X_{011}
♡♣♣ $X_{111} + X_{110} + X_{101} + X_{000}$
(inputPerm, 1, (1 2), id)
♣♣♡ $X_{100} + X_{010} + X_{001} + X_{(\mathsf{id},\mathsf{id})10} + X_{(\mathsf{id},\mathsf{id})01} + X_{((2\,3),(1\,2))00}$
♣♡♣ $X_{000} + X_{(\mathsf{id},\mathsf{id})11} + X_{((2\,3),(1\,2))11} + X_{((2\,3),(1\,2))10} + X_{((2\,3),(1\,2))01}$
♡♣♣ $X_{111} + X_{110} + X_{101} + X_{011} + X_{(\mathsf{id},\mathsf{id})00}$
(result, 1)
✓

Figure 11 State diagram of the three-inputs majority protocol from [34]. We track the case that player 1 is an active attacker who may make incoherent choices during his two inputPerm actions.

Finally, we discuss protocols where input is given by the players via choosing a permutation, including a corresponding adaptation of the state tree formalism, and present active attacks on two majority protocols from the literature.

Open Problems

Some card-minimal protocols, e.g. the general k-ary boolean function protocol of [12], use non-closed shuffles, with no evidence yet that this is necessary. As we have determined that uniform closed shuffles are a natural shuffle class, which can be done actively secure, it is interesting to find card-minimal protocols using only uniform closed shuffles.

Another natural problem is to implement more general shuffles, and even to characterize the shuffles which are possible with (a linear number of) helping cards, and the assumption of the security of a uniform random cut. To give one non-trivial example, we show in Appendix D in the full version [10] how any subset of a cut can be implemented.

References

1 Yuta Abe, Yu ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Five-card and protocol in committed format using only practical shuffles. In Keita Emura, Jae Hong Seo, and Yohei Watanabe, editors, *APKC@AsiaCCS 2018*, pages 3–8. ACM, 2018. doi:10.1145/3197507.3197510.

2 Eddie Cheung, Chris Hawthorne, and Patrick Lee. CS 758 project: Secure computation with playing cards, 2013. URL: https://cdchawthorne.com/writings/secure_playing_cards.pdf.

3 Claude Crépeau and Joe Kilian. Discreet solitary games. In Douglas R. Stinson, editor, *CRYPTO '93*, volume 773 of *LNCS*, pages 319–330. Springer, 1993. doi:10.1007/3-540-48329-2_27.

4 Bert den Boer. More efficient match-making and satisfiability: The five card trick. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT '89*, volume 434 of *LNCS*, pages 208–217. Springer, 1989. doi:10.1007/3-540-46885-4_23.

5 Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003. doi:10.1007/978-3-540-45146-4_27.

6 Rie Ishikawa, Eikoh Chida, and Takaaki Mizuki. Efficient card-based protocols for generating a hidden random permutation without fixed points. In Cristian S. Calude and Michael J. Dinneen, editors, *UCNC 2015*, volume 9252 of *LNCS*, pages 215–226. Springer, 2015. doi:10.1007/978-3-319-21819-9_16.

7 Julia Kastner, Alexander Koch, Stefan Walzer, Daiki Miyahara, Yu ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. The minimum number of cards in practical card-based protocols. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017*, volume 10626 of *LNCS*, pages 126–155. Springer, 2017. doi:10.1007/978-3-319-70700-6_5.

8 Alexander Koch. The landscape of optimal card-based protocols. *IACR Cryptology ePrint Archive*, 2018. Report 2018/951. URL: https://eprint.iacr.org/2018/951.

9 Alexander Koch. *Cryptographic Protocols from Physical Assumptions*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2019. doi:10.5445/IR/1000097756.

10 Alexander Koch and Stefan Walzer. Foundations for actively secure card-based cryptography. *IACR Cryptology ePrint Archive*, 2017. Report 2017/423. URL: https://eprint.iacr.org/2017/423.

11 Alexander Koch and Stefan Walzer. Private function evaluation with cards. *IACR Cryptology ePrint Archive*, 2018. Report 2018/1113. URL: https://eprint.iacr.org/2018/1113.

12 Alexander Koch, Stefan Walzer, and Kevin Härtel. Card-based cryptographic protocols using a minimal number of cards. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9452 of *LNCS*, pages 783–807. Springer, 2015. doi:10.1007/978-3-662-48797-6_32.

13 Antonio Marcedone, Zikai Wen, and Elaine Shi. Secure dating with four or fewer cards. *IACR Cryptology ePrint Archive*, 2015. Report 2015/1031. URL: https://eprint.iacr.org/2015/1031.

14 Takaaki Mizuki. Card-based protocols for securely computing the conjunction of multiple variables. *Theoretical Computer Science*, 622:34–44, 2016. doi:10.1016/j.tcs.2016.01.039.

15 Takaaki Mizuki, Isaac Kobina Asiedu, and Hideaki Sone. Voting with a logarithmic number of cards. In Giancarlo Mauri, Alberto Dennunzio, Luca Manzoni, and Antonio E. Porreca, editors, *UCNC 2013*, volume 7956 of *LNCS*, pages 162–173. Springer, 2013. doi:10.1007/978-3-642-39074-6_16.

16 Takaaki Mizuki, Michihito Kumamoto, and Hideaki Sone. The five-card trick can be done with four cards. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 598–606. Springer, 2012. doi:10.1007/978-3-642-34961-4_36.

17 Takaaki Mizuki and Hiroki Shizuya. A formalization of card-based cryptographic protocols via abstract machine. *International Journal of Information Security*, 13(1):15–23, 2014. doi:10.1007/s10207-013-0219-4.

18 Takaaki Mizuki and Hiroki Shizuya. Practical card-based cryptography. In Alfredo Ferro, Fabrizio Luccio, and Peter Widmayer, editors, *FUN 2014*, volume 8496 of *LNCS*, pages 313–324. Springer, 2014. doi:10.1007/978-3-319-07890-8_27.

19 Takaaki Mizuki and Hideaki Sone. Six-card secure AND and four-card secure XOR. In Xiaotie Deng, John E. Hopcroft, and Jinyun Xue, editors, *FAW 2009*, volume 5598 of *LNCS*, pages 358–369. Springer, 2009. doi:10.1007/978-3-642-02270-8_36.

20 Tal Moran and Moni Naor. Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 88–108. Springer, 2006. doi:10.1007/11761679_7.

21 Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. *Theoretical Computer Science*, 411(10):1283–1310, 2010. doi:10.1016/j.tcs.2009.10.023.

22 Takeshi Nakai, Satoshi Shirouchi, Mitsugu Iwamoto, and Kazuo Ohta. Four cards are sufficient for a card-based three-input voting protocol utilizing private permutations. In Junji Shikata, editor, *ICITS 2017*, volume 10681 of *LNCS*, pages 153–165. Springer, 2017. doi:10.1007/978-3-319-72089-0_9.

23 Takeshi Nakai, Yuuki Tokushige, Yuto Misawa, Mitsugu Iwamoto, and Kazuo Ohta. Efficient card-based cryptographic protocols for millionaires' problem utilizing private permutations. In Sara Foresti and Giuseppe Persiano, editors, *CANS 2016*, volume 10052 of *LNCS*, pages 500–517, 2016. doi:10.1007/978-3-319-48965-0_30.

24 Valtteri Niemi and Ari Renvall. Secure multiparty computations without computers. *Theoretical Computer Science*, 191(1-2):173–183, 1998. doi:10.1016/S0304-3975(97)00107-2.

25 Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Card-based protocols for any boolean function. In Rahul Jain, Sanjay Jain, and Frank Stephan, editors, *TAMC 2015*, volume 9076 of *LNCS*, pages 110–121. Springer, 2015. doi:10.1007/978-3-319-17142-5_11.

26 Takuya Nishida, Takaaki Mizuki, and Hideaki Sone. Securely computing the three-input majority function with eight cards. In Adrian Horia Dediu, Carlos Martín-Vide, Bianca Truthe, and Miguel A. Vega-Rodríguez, editors, *TPNC 2013*, volume 8273 of *LNCS*, pages 193–204. Springer, 2013. doi:10.1007/978-3-642-45008-2_16.

27 Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. An implementation of non-uniform shuffle for secure multi-party computation. In *AsiaPKC 2016*, pages 49–55. ACM, 2016. doi:10.1145/2898420.2898425.

28 Akihiro Nishimura, Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Five-card secure computations using unequal division shuffle. In Adrian Horia Dediu, Luis Magdalena, and Carlos Martín-Vide, editors, *TPNC 2015*, volume 9477 of *LNCS*, pages 109–120. Springer, 2015. doi:10.1007/978-3-319-26841-5_9.

29 Akihiro Nishimura, Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Card-based protocols using unequal division shuffles. *Soft Computing*, 22(2):361–371, 2018. doi:10.1007/s00500-017-2858-2.

30 Kazumasa Shinagawa, Takaaki Mizuki, Jacob C. N. Schuldt, Koji Nuida, Naoki Kanayama, Takashi Nishide, Goichiro Hanaoka, and Eiji Okamoto. Secure multi-party computation using polarizing cards. In Keisuke Tanaka and Yuji Suga, editors, *IWSEC 2015*, volume 9241 of *LNCS*, pages 281–297. Springer, 2015. doi:10.1007/978-3-319-22425-1_17.

31 Anton Stiglic. Computations with a deck of cards. *Theoretical Computer Science*, 259(1-2):671–678, 2001. doi:10.1016/S0304-3975(00)00409-6.

32 Itaru Ueda, Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. How to implement a random bisection cut. In Carlos Martín-Vide, Takaaki Mizuki, and Miguel A. Vega-Rodríguez, editors, *TPNC 2016*, pages 58–69. Springer, 2016. doi:10.1007/978-3-319-49001-4_5.

33 Tom Verhoeff. The zero-knowledge match maker, 2014. URL: https://www.win.tue.nl/~wstomv/publications/liber-AMiCorum-arjeh-bijdrage-van-tom-verhoeff.pdf.

34 Yohei Watanabe, Yoshihisa Kuroki, Shinnosuke Suzuki, Yuta Koga, Mitsugu Iwamoto, and Kazuo Ohta. Card-based majority voting protocols with three inputs using three cards. In *International Symposium on Information Theory and Its Applications, ISITA 2018*, pages 218–222. IEEE, 2018. doi:10.23919/ISITA.2018.8664324.

Hyperbolic Minesweeper Is in P

Eryk Kopczyński
Institute of Informatics, University of Warsaw, Poland
http://www.mimuw.edu.pl/~erykk/
erykk@mimuw.edu.pl

Abstract

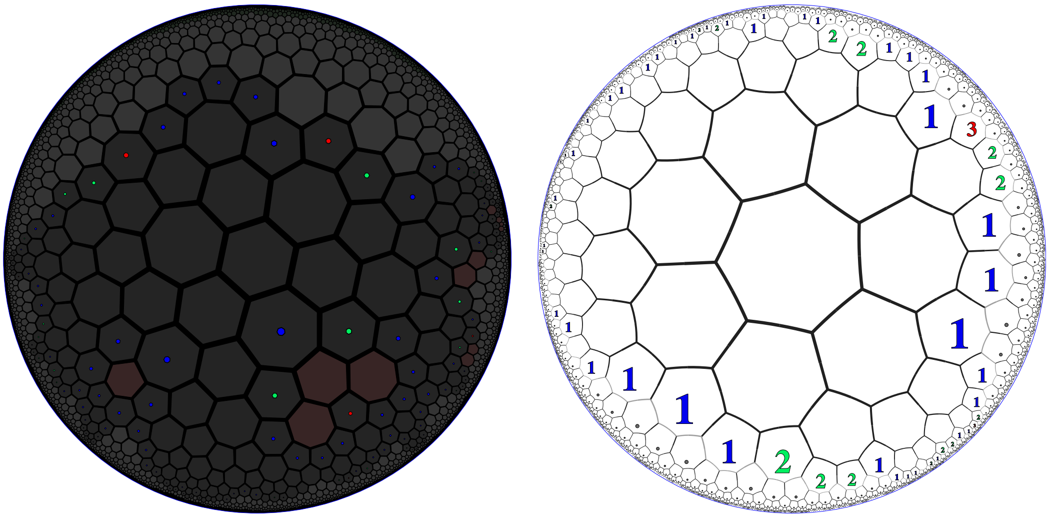
We show that, while Minesweeper is NP-complete, its hyperbolic variant is in P. Our proof does not rely on the rules of Minesweeper, but is valid for any puzzle based on satisfying local constraints on a graph embedded in the hyperbolic plane.

2012 ACM Subject Classification Theory of computation → Representations of games and their complexity; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Minesweeper

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.18

1 Introduction



(a) graphical mode, bitruncated tessellation. **(b)** text mode, regular tessellation.

Figure 1 Hyperbolic Minesweeper [7]. In (a), the default settings are used (bitruncated order-3 heptagonal tessellation, numbers of adjacent mines are color-coded; some of the mines that the players is sure of are marked red). In (b) we play on an order-3 heptagonal tessellation, and numbers are shown.

Minesweeper is a popular game included with many computer systems; it also exists in the puzzle form. In the puzzle form, every cell in a square grid either contains a number or is empty. The goal of the puzzle is to assign mines to the empty squares in such a way that every number n is adjacent (orthogonally or diagonally) to exactly n mines.

This puzzle is a well-known example of a NP-complete problem [6]. Its popularity has also spawned many variants played on different grids, from changing the tessellation of the plane, to changing the number of dimensions (six-dimensional implementations of Minesweeper exist) to changing the underlying geometry.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 18; pp. 18:1–18:7
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we will be playing Minesweeper on a tessellation of the hyperbolic plane. Minefield, a land based on hyperbolic Minesweeper is available in HyperRogue [7]; this implementation differs from the standard Minesweeper in multiple ways, e.g., by being played on an infinite board, however, the basic idea is the same. Another implementation is Warped Mines for iOS [14], which has the same rules as the usual Minesweeper except the board, which is a bounded subset of the order-3 heptagonal tiling of the hyperbolic plane.

From the point of view of a computer scientist, the most important distinctive property of hyperbolic geometry is exponential growth: the area of a circle of radius r grows exponentially with r. It is also more difficult to understand than Euclidean geometry. While these properties often cause computational geometry problems to be more difficult, it also gives hyperbolic geometry applications in data visualization [8, 10] and data analysis [11].

In this paper we show that hyperbolic geometry makes Minesweeper easier: the hyperbolic variant of Minesweeper is in P. Our proof will not rely on the specific rules of Minesweeper nor work with any specific tessellation of the hyperbolic plane; instead, it will work with any puzzle based on satisfying local constraints, on any graph that naturally embeds into the hyperbolic plane.

2 Hyperbolic Geometry

We denote the hyperbolic plane with $\mathbb{H}^2$. Since this paper requires only the basic understanding of hyperbolic geometry, we will not include the formal definition; see [2], or [7] for an intuitive introduction. Figures 1 and 2 shows the hyperbolic plane in the Poincaré disk model, which is a projection which represents angles faithfully, but not the distances: the scale gets smaller and smaller as we get closer to the circle bounding the model. In particular, all the heptagons in Figure 1b are the same size, all the heptagons in Figure 1a are the same size, and all the hexagons in Figure 1a are the same size. We denote the distance between two points $x, y \in \mathbb{H}^2$ by $\delta(x, y)$. The set of points in distance at most r from x is denoted with $B(x, r)$. The area of $B(x, r)$ is $2\pi(\cosh r - 1)$, which we denote with $\operatorname{area}(r)$. The perimeter of $B(x, r)$ is $2\pi \sinh r$, which we denote with $\operatorname{peri}(r)$. Note that both area and peri grow exponentially: $\operatorname{area}(r) = \Theta(e^r)$ and $\operatorname{peri}(r) = \Theta(e^r)$.

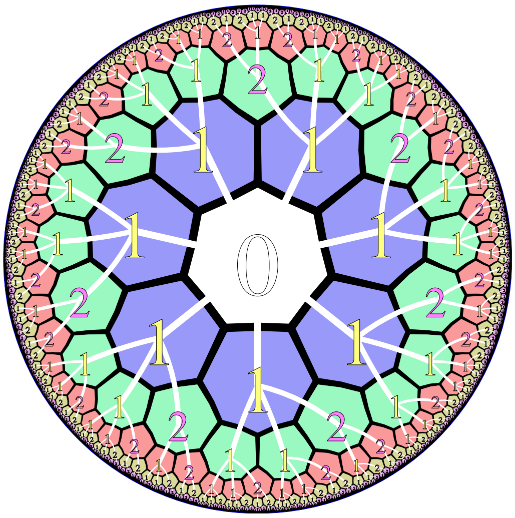


Figure 2 Exponential growth in the hyperbolic plane.

We include Figure 2 as a simple introduction to the structure of the hyperbolic plane and its exponential growth. This is the order-3 heptagonal tessellation [9]. The numbers correspond to the number of adjacent tiles which are closer to the center tile. This tessellation can be constructed as follows. We construct a tree of tiles: the central tile has seven children of type 1; each tile of tile 1 has two children of type 1, and one child of type 2; each tile of tile 2 has one child of type 1 and one child of type 2. To produce the actual tessellation, we also connect adjacent tiles on the same level, and to the leftmost child of the tile on the right. The hyperbolic plane can be seen as a continuous version of the graph obtained using this construction. Since every tile has at least two children, there are exponentially many tiles in r steps from the central tile.

3 Hyperbolic Graph

We need a suitable definition of a hyperbolic graph. The obvious definition, obtained by changing "plane" to "hyperbolic plane" in the definition of a planar graph, is equivalent to the definition of the planar graph (the plane and the hyperbolic plane are homeomorphic). Another definition found in literature is the notion of Gromov δ-hyperbolic graph [1]; this definition is based on the observation that, in the hyperbolic plane, triangles are thin, i.e., for any three vertices a, b, c, every point on any shortest path from a to c must be in distance at most δ from either the shortest path from a to b, or the shortest path from b to c. The parameter δ measures tree-likeness of the graph (it can be easily seen that trees are 0-hyperbolic). However, this definition is not suitable for us, because every graph G becomes 2-hyperbolic when we add a vertex v_* connected to every $v \in V(G)$; our main result is not true for such graphs. We propose the following definition:

▶ **Definition 1.** *A (r,d)-**hyperbolic graph** is a graph $G = (V,E)$ such that there exists an embedding $m : V \to \mathbb{H}^2$ such that:*

- *for $v_1 \neq v_2$, $\delta(m(v_1), m(v_2)) > r$,*
- *for $\{v_1, v_2\} \in E$, $\delta(m(v_1), m(v_2)) < d$,*
- *if we draw every edge $\{v_1, v_2\} \in E$ as a straight line segment between $m(v_1)$ and $m(v_2)$, these edges do not cross, nor they get closer to vertices other than v_1 or v_2 than $r/2$.*

Intuitively, $r/2$ is the radius of every vertex; this parameter bounds the density of our graph embedding. The parameter d gives the maximum distance between two vertices connected with an edge.

This definition includes finite subsets of all regular and semiregular hyperbolic tessellations (such as the ones shown in Figure 1). We denote $N(v)$ to be the neighborhood of $v \in V$, i.e., $\{v\} \cup \{w \in V : \{v,w\} \in E\}$. (Our proof also works with neighborhoods of larger radius.)

Remark. All (r,d)-hyperbolic graphs have degree bounded by a constant (for fixed r,d). Indeed, let $v \in V$. For every $w \in N(v)$, the circles $B(w, \frac{r}{2})$ are disjoint, and they fit in $B(v, d + \frac{r}{2})$. Therefore, $|N| \leq \text{area}(d + \frac{r}{2})/\text{area}(\frac{r}{2})$.

4 Hyperbolic Local Constraint Satisfaction Problem

Below we state our main result.

▶ **Theorem 2.** *Fix the set of colors K and the parameters (r,d). The following problem (Hyperbolic Local Constraint Satisfaction Problem, HLCSP) can be solved in polynomial time:*

INPUT:

- *a (r,d)-hyperbolic graph $G = (V,E)$;*
- *for every vertex $v \in V$, $m(v)$, a subset of $K^{N(v)}$.*

OUTPUT: *Is there a coloring $c : V \to K$ such that for every $v \in V$, $c_{|N(v)} \in m(v)$?*

HLCSP generalizes hyperbolic minesweeper. We have two colors (no mine and mine), and for every $v \in V$ containing a number k, the constraint $m(v)$ says that v contains no mine itself, and exactly k vertices in $N(v)$ contain a mine.

To prove Theorem 2, it is enough to prove that the following problem (HECSP) is in P.

▶ **Theorem 3.** *HLCSP reduces to the Hyperbolic Edge Constraint Satisfaction Problem (HECSP) given as follows:*

INPUT:

- *a (r, d)-hyperbolic graph $G = (V, E)$;*
- *for every edge $e \in E$, $m(e)$, a subset of K^e.*

OUTPUT: *Is there a coloring $c : V \to K$ such that for every $e \in E$, $c_{|e} \in m(e)$?*
(Note: this reduction changes the set of admissible colors K.)

Proof. Let (G, m) be the instance of HLCSP. We will be coloring V using colors $K' = \{1, \ldots, k\}$, where k is the greatest number of elements of $m(v)$ (since (r, d)-hyperbolic graphs have bounded degree and K is fixed, k is also bounded). Enumerate every elements of $m(v)$ with one color from K'. For $e = \{v_1, v_2\} \in E$, $m(e)$ is the set of all colorings $c : \{v_1, v_2\} \to K$ which are consistent, i.e., k_1 denotes $c_1 \in m(v_1)$ and k_2 denotes $c_2 \in m(v_2)$, and c_1 equals c_2 on all the vertices in $N(v_1) \cap N(v_2)$. ◀

5 Proof of Theorem 2

We will be using the following result [12, 3]:

▶ **Theorem 4.** *Given a planar graph $G = (V, E)$ and a number t, it is possible to either find a $t \times t$ grid as a minor of G, or produce a tree decomposition of G of width $\leq 5t - 6$, in time $O(n^2 \log(n))$, where $n = |V|$.*

▶ **Definition 5.** *A tree decomposition of width w of a graph $G = (V, E)$ is (V_T, E_T, X) where (V_T, E_T) is a tree, and $X : V_T \to P(V)$ is a mapping which assigns a subset of V of cardinality at most $w + 1$ to every vertex in V_T, such that:*

- *For every $v \in V$, the set of vertices $b \in V_T$ such that $v \in X_b$ is connected,*
- *For every $e \in E$, there exists a $b \in V_T$ such that $e \subseteq X_b$.*

We can assume that our tree is rooted in $r \in V_T$. For $b \in V_T$, let X_b^+ be the union of $X_{b'}$ for all b' which are descendants of b.

▶ **Lemma 6.** *Without loss of generality we can assume that every $b \in V_T$ falls into one of the following cases:*

- *b is a leaf and $|X_b| = 1$,*
- *b has a single child b' and $X_{b'} = X_b \cup \{v\}$,*
- *b has a single child b' and $X_{b'} = X_b - \{v\}$,*
- *b has two children b_1 and b_2, $X_b = X_{b_1} = X_{b_2}$, and $X_{b_1}^+ - X_b$ and $X_{b_2}^+ - X_b$ are non-empty and disjoint.*

▶ **Theorem 7.** *If a (r, d)-hyperbolic graph G contains a $t \times t$ grid as a minor, then $t = O(\log(V(G))$.*

Proof. Without loss of generality we can assume $t = 2k + 1$. Such a $t \times t$ grid contains k cycles around the center of the grid. Let $v_c \in V$ be the vertex corresponding to this center. We have k cycles $C_1, \ldots, C_k$ in the graph V, each of which surrounds v_c and the preceding cycles. We use the following lemma:

▶ **Lemma 8.** *Every point in the drawing of cycle C_i is in distance $\Omega(i)$ from v. (The drawing is the polygon obtained as a union of the edges embedded in $\mathbb{H}^2$.)*

Therefore, $|C_i| \geq \mathrm{peri}(\Omega(i))/d$. Since peri grows exponentially, we get $k = O\log(|V|)$. ◀

Proof of Lemma 8. We will show that if every point in the drawing of C_i is in distance at least x from v, then every point in the drawing of C_{i+1} is in distance $x + c$ from v. By induction, this is enough to prove Lemma 8.

Figure 3 shows what happens for an Euclidean graph. (We define Euclidean graph just like hyperbolic graph except the differing underlying geometry.) In this picture, the drawing of C_i is a hexagon; cycle C_i has six vertices around v_c. For each of these seven vertices the exclusion zone of radius $r/2$ around it is shown (in green). It is clear from the picture (and the definition of Euclidean graph) that no point in the drawing of C_{i+1} may fall into the gray/green region. All points in distance $\leq r/2$ from C_i fall in the gray/green region, thus our claim is true for $c = h = r/2$.

In the hyperbolic plane the situation is slightly different. Because parallel lines work differently in the hyperbolic plane (they "diverge"), we have $c = h < r/2$, where h depends on r and d (we have $h = \Theta(re^{-d})$ for large values of d). Still, our claim holds with $c > 0$. ◀

▶ **Corollary 9.** *Given a planar graph $G = (V, E)$, it is possible to find a tree decomposition of width $O(\log |V|)$ in polynomial time.*

Proof. From Theorem 7 choose $t = O(\log |V|)$ such that G does not contain a $k \times k$ grid. From Theorem 4, it is possible to find a tree decomposition of width $5t - 6 = O(\log |V|)$. ◀

▶ **Corollary 10.** *HECSP (and thus HLCSP) can be solved in polynomial time.*

Proof. From Corollary 9 we can find a tree decomposition (V_T, E_T, B) of width $w = O(\log |V|)$. Then we use Dynamic Programming over (V_T, E_T). For every $b \in V_T$, we compute $s(w)$, the set of all possible colorings $c : X_b \in K$ such that there exists a coloring $c : X_b^+$ which extends c and which satisfies all the constraints on edges in X_b^+. This can be computed straightforwardly in every case from Lemma 6. The whole algorithm works in $O(|V| \cdot |K|^w)$, which is polynomial in $|V|$. ◀

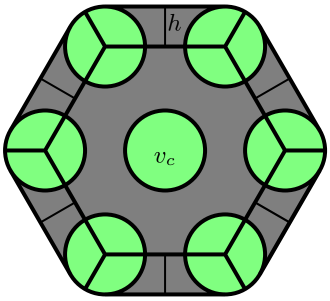
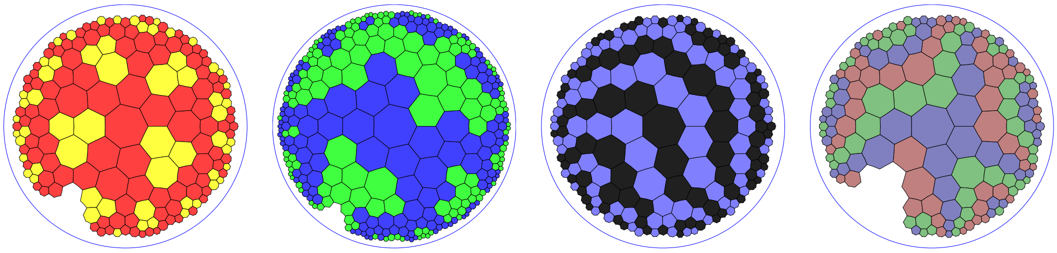


Figure 3 Cycles around v_c.



(a) two yellow. **(b)** one group. **(c)** two groups. **(d)** four groups.

Figure 4 Random colorings satisfying various constraints. Since the degree of our polynomial is quite high, these pictures took about a minute to make. Some cells have been removed from the full disk to reduce the treewidth.

6 Conclusion

We have shown that Minesweeper on hyperbolic tessellations can be solved in polynomial time. Our method is general: it works for any (r, d)-hyperbolic graph, and for any problem based on satisfying local constraints. Other than solving puzzles, this may be applied to procedural content generation. For example, the Wave Function Collapse (WFC) algorithm [4, 5] is used in procedurally generated games to procedurally generate maps satisfying local constraints (e.g., a mountain should not appear close to ocean, or roads should not branch nor end abruptly). In general finding out whether such constraints can be satisfied is NP-complete (although this does not happen in typical PCG applications). Our algorithm can be adjusted to count the number of satisfying colorings, and to produce one of them, randomly chosen (Figure 4).

Our results do not generalize to the three-dimensional hyperbolic space $\mathbb{H}^3$. This is because the Euclidean plane can be isometrically embedded in the three-dimensional hyperbolic space $\mathbb{H}^3$ as a horosphere. Minesweeper played on a tiling which tessellates such a horosphere into squares is NP-complete.

HyperRogue also lets the player play Minesweeper in bounded hyperbolic manifolds, such as the Klein quartic or other Hurwitz manifolds. Hurwitz manifolds are quotient spaces of $\mathbb{H}^2$ which can be tiled with regular heptagons with angles 120° (see Figure 1b), and that are maximally symmetric, i.e., there exists an isometry of the Hurwitz manifold M into itself which map any heptagon with any orientation into any heptagon with any orientation. Tessellations of quotient space are no longer planar, thus Theorem 4 nor our proof of Theorem 7 is no longer valid in quotient spaces of $\mathbb{H}^2$. Therefore, the complexity of Minesweeper on such manifolds does not follow from our results. There are less symmetric hyperbolic manifolds into which the Euclidean square grid, or even higher-dimensional Euclidean grids, can be embedded [13], and thus Minesweeper on them is NP-complete; however, highly symmetric manifolds have a more restricted structure.

References

1 Sergio Bermudo, José M. Rodríguez, José M. Sigarreta, and Jean-Marie Vilaire. Gromov hyperbolic graphs. *Discrete Mathematics*, 313(15):1575–1585, 2013. doi:10.1016/j.disc.2013.04.009.

2 James W. Cannon, William J. Floyd, Richard Kenyon, Walter, and R. Parry. Hyperbolic geometry. In *In Flavors of geometry*, pages 59–115. University Press, 1997. Available online at http://www.msri.org/communications/books/Book31/files/cannon.pdf.

3 Alexander Grigoriev. Tree-width and large grid minors in planar graphs. *Discrete Mathematics & Theoretical Computer Science*, 13:13–20, January 2011.

4 Maxim Gumin. WaveFunctionCollapse, 2017. URL: https://github.com/mxgmn/WaveFunctionCollapse.

5 Isaac Karth and Adam M. Smith. WaveFunctionCollapse is Constraint Solving in the Wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3102071.3110566.

6 Richard Kaye. Minesweeper is np-complete. *The Mathematical Intelligencer*, 22(2):9–15, March 2000. doi:10.1007/BF03025367.

7 Eryk Kopczyński, Dorota Celińska, and Marek Čtrnáct. Hyperrogue: Playing with hyperbolic geometry. In David Swart and Carlo H. Séquin and Kristóf Fenyvesi, editor, *Proceedings of Bridges 2017: Mathematics, Art, Music, Architecture, Education, Culture*, pages 9–16, Phoenix, Arizona, 2017. Tessellations Publishing. Available online at http://archive.bridgesmathart.org/2017/bridges2017-9.pdf.

8 John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. doi:10.1145/223904.223956.

9 Maurice Margenstern. Pentagrid and heptagrid: the fibonacci technique and group theory. *Journal of Automata, Languages and Combinatorics*, 19(1-4):201–212, 2014. doi:10.25596/jalc-2014-201.

10 Tamara Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Computer Graphics and Applications*, 18(4):18–23, 1998. doi:10.1109/38.689657.

11 Fragkiskos Papadopoulos, Maksim Kitsak, M. Angeles Serrano, Marian Boguñá, and Dmitri Krioukov. Popularity versus Similarity in Growing Networks. *Nature*, 489:537–540, September 2012.

12 N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.

13 Zeno Rogue. HyperRogue: Experiments with Geometry, 2020. URL: http://www.roguetemple.com/z/hyper/geoms.php.

14 Sci-Tech Binary Ltd. Co. Warped mines, 2019. URL: https://apps.apple.com/br/app/hypersweeper/id1450066199.

Train Tracks with Gaps

William Kuszmaul
Massachusetts Institute of Technology, CSAIL, Cambrige, MA, USA
kuszmaul@mit.edu

Abstract

We identify a tradeoff curve between the number of wheels on a train car, and the amount of track that must be installed in order to ensure that the train car is supported by the track at all times. The goal is to build an elevated track that covers some large distance ℓ, but that consists primarily of gaps, so that the total amount of feet of train track that is actually installed is only a small fraction of ℓ. In order so that the train track can support the train at all points, the requirement is that as the train drives across the track, at least one set of wheels from the rear quarter and at least one set of wheels from the front quarter of the train must be touching the track at all times.

We show that, if a train car has n sets of wheels evenly spaced apart in its rear and n sets of wheels evenly spaced apart in its front, then it is possible to build a train track that supports the train car but uses only $\Theta(\ell/n)$ feet of track. We then consider what happens if the wheels on the train car are not evenly spaced (and may even be configured adversarially). We show that for any configuration of the train car, with n wheels in each of the front and rear quarters of the car, it is possible to build a track that supports the car for distance ℓ and uses only $O\left(\frac{\ell \log n}{n}\right)$ feet of track. Additionally, we show that there exist configurations of the train car for which this tradeoff curve is asymptotically optimal. Both the upper and lower bounds are achieved via applications of the probabilistic method.

The algorithms and lower bounds in this paper provide simple illustrative examples of many of the core techniques in probabilistic combinatorics and randomized algorithms. These include the probabilistic method with alterations, the use of McDiarmid's inequality within the probabilistic method, the algorithmic Lovász Local Lemma, the min-hash technique, and the method of conditional probabilities.

2012 ACM Subject Classification Theory of computation

Keywords and phrases probabilistic method, algorithms, trains, Lovász Local Lemma, McDiarmid's Inequality

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.19

Funding Funded by a Fannie and John Hertz Fellowship and an NSF GRFP fellowship. Research also was sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Acknowledgements The author would like to thank Michael A. Bender, Bradley C. Kuszmaul, and Charles E. Leiserson for several useful conversations about train tracks. The author would also like to thank Jake Hillard for offering his engineering expertise, and observing that train tracks with an asymptotically small number of pillars would likely encounter practical difficulties in the real world.

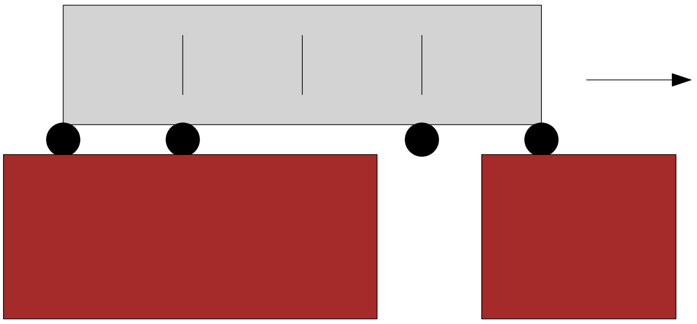
A Gap in the Track

A few years ago, while traveling on a train, and on only a few hours of sleep, I was staring out the window. The train crossed a bridge over a road, and the ground was momentarily replaced by a steep drop. Startled, my sleep-deprived mind briefly wondered whether there was still a track underneath us. *Of course there is,* I thought to myself. *Without a track, the train car would have fallen into the gap.*

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 19; pp. 19:1–19:11
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ah, no so fast! responded the latent mathematician inside me. *If the train car had more than two sets of wheels, then perhaps it could cross the bridgeless gap without falling in.* It was true.

Consider, for example, a train with four sets of wheels: one set in the rear, one set in the front, and one set in each of the first and third quartiles.



As long as the gap in the track is less than the distance between any pair of wheels, then at least three sets of wheels will touch track at all times. Assuming that the center of mass of the train is in the middle half of the train, it follows that the train does not fall into the gap.

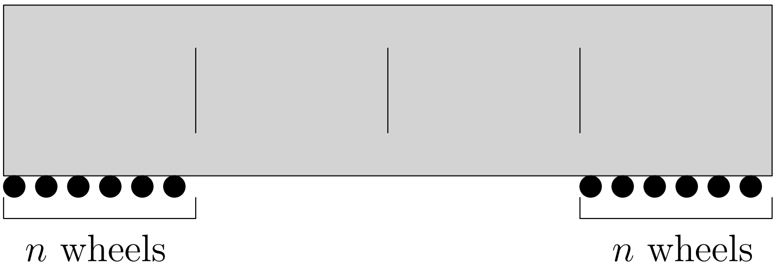
In fact, continued the latent mathematician, *what if we have n sets of wheels? Maybe we can build a mono-rail using an asymptotically small amount of track.*

That's a stupid idea, responded I. *Gaps in train track are not something to optimize.*

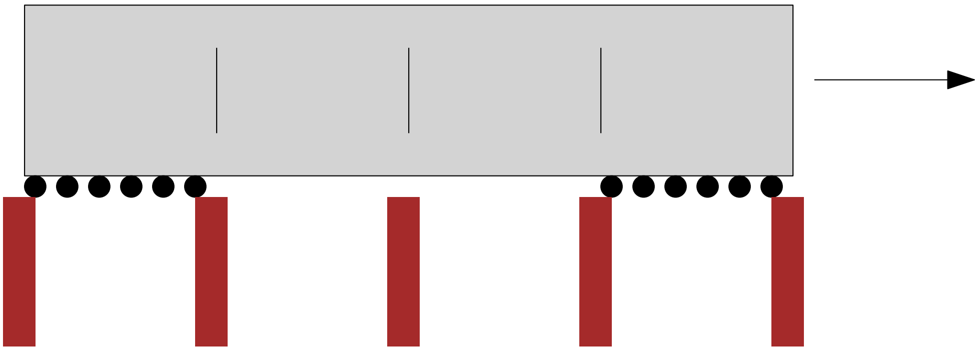
But I was sleep deprived, so I did it anyway.

The Basic Observation: More Wheels Means Less Track

Consider a train car with $2n$ sets of wheels. Half the sets are evenly dispersed across the first quarter of the train car, and half the sets are evenly dispersed across the final quarter. The train can safely drive down the track as long as at least one set of wheels from each side of the train car is touching track at all times. The train car looks something like this:



Want to build a monorail, but you're short on track? No problem! You can get away with filling in only an $O(1/n)$ fraction of the track:



Every fourth of a train length, we place a piece of track whose length is a $\frac{1}{4n}$ fraction of the length of the train car. We get asymptotic cost savings!

To see that this is the best we can do, suppose that the fraction of track that is filled in is less than $\frac{1}{n}$, and for symmetry sake suppose the track is circular (i.e., the end of the track loops back to the start). If we place the train at a random position in the circular track, then each wheel has a less than $\frac{1}{n}$ chance of touching track. By a union bound, it follows that the probability of any wheel in the rear quarter of the train touching the track is less than 1. Thus no matter how the track is placed, if the total fraction of track that is filled in is less than $\frac{1}{n}$, then there is some position at which the train falls through.

Paper Outline

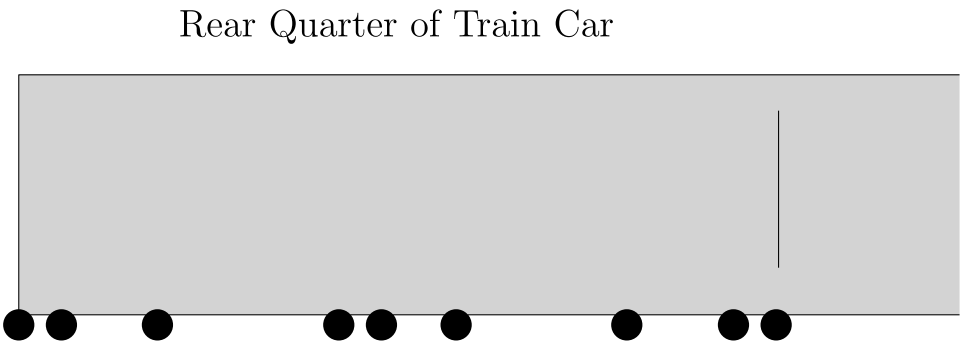
The rest of the paper considers the situation in which the wheels on the train car are placed unevenly (and possibly even adversarially!) in each of the front and rear quarters of the car. Section 1 describes the problem in more detail, and shows that for any configuration of the train car, with n wheels in each of the front and rear quarters of the car, it is possible to efficiently build a track that supports the car for distance ℓ and uses only $O\left(\frac{\ell \log n}{n}\right)$ feet of track. Section 2 then establishes a matching lower bound, showing that there exist configurations of the train car for which $\Omega\left(\frac{\ell \log n}{n}\right)$ feet of track are required. Both the upper and lower bounds are achieved via applications of the probabilistic method.

The train-track problem serves as a veritable playground for applying many of the core techniques from probabilistic combinatorics and randomized algorithms to a simple and fun problem. Section 3 give three alternative algorithms for achieving the upper bound of $O\left(\frac{\ell \log n}{n}\right)$, each of which builds on a different technique.

Combined, the algorithms and lower bounds in this paper give simple illustrative examples of the algorithmic Lovász Local Lemma, the min-hash technique, the method of conditional probabilities, the probabilistic method with alterations, and the use of McDiarmid's inequality within the probabilistic method.

1 Train Cars with Arbitrary Wheel Arrangements

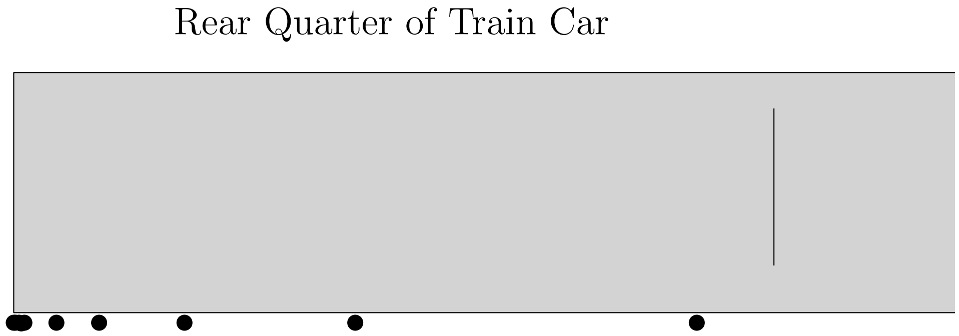
Consider a train car that has n wheels in its rear quarter and n wheels in its front quarter, but suppose that the wheels *aren't* evenly spaced. For example, maybe the rear of the car looks something like this:



Can we still fill in an asymptotically small fraction of the track in a way that will allow the train car to drive down the track? In other words, can we place down a small amount of track in a way so that, as the train drives down it there is always at least one wheel from each of the front and rear quarters of the train touching track? It turns out that, via a simple application of the probabilistic method with alterations, we can.

To formalize the situation, let's focus just on the first quarter of the train. (In particular, up to a constant factor in the amount of train track that we install, we can consider the two quarters of the train separately.) Suppose this portion of the train is f feet long, and assume that each of the n sets of wheels resides at a distinct integer offset from the rear of the train. Our goal is to build a train track that is ℓ feet long. We build the train track out of ***pillars*** that are each 1 foot long and are each placed at integer positions on the track. We are required to put down the pillars in a way so that, as the train drives down the track, at least one wheel from the rear quarter is always touching the track (i.e., touching some pillar). We want to use as little track as possible, with the best we could hope for being a total of $\frac{\ell}{n}$ pillars.

As a reminder, there are three variables: the number of wheels n (in the quarter of the train car that we're considering), the length f of one quarter of the train car, and the length ℓ of the train track. In general, we have $n \leq f \leq \ell$. Note that, although n and f could reasonably be close to one another (e.g., $f = 2n$), we also want to be able to handle cases where $n \ll f$. This allows for the train car to be configured in truly strange ways – for example, the positions of the wheels could even form a geometric series:



1.1 A Randomized Algorithm for Building Track

Our algorithm is a simple example of the probabilistic method with alterations. In particular, we begin by randomly constructing a track that uses only a small number of pillars, and then we show that this track can be slightly altered in order to support the rear of the train car at every point.

We begin by installing each pillar randomly with probability $\frac{\ln n}{n}$. Even though this strategy has nothing to do with the structure of where the wheels are on the train, it already does remarkably well. In particular, if we place the train at some given position on the track, then there are n different pillar positions that could potentially hold up the rear quarter. Each of these pillars positions has a $\frac{\ln n}{n}$ probability of having a pillar installed. It follows that, for a given position on the track, the rear quarter of the train has a

$$\left(1 - \frac{\ln n}{n}\right)^n$$

probability of falling through the track. Taking advantage of the identity $\left(1 - \frac{1}{k}\right)^k \le \frac{1}{e}$, which holds for any $k \ge 1$, it follows that the wheels fall through the track with probability at most,

$$\frac{1}{e^{\ln n}} = \frac{1}{n}.$$

In other words, out of all the places we can place the train on the track, only a $\frac{1}{n}$-fraction of them will be problematic (in expectation). To fix this, we can just install one additional pillar to remedy each of these problematic positions. The result is a train track that fully supports the rear quarter of our train, and that uses only $\left(\frac{1+\ln n}{n}\right)\ell$ total feet of track, in expectation.

Of course, this isn't quite as good as we did when the wheels were evenly spaced out (we are a roughly $\ln n$ factor worse). But it's still pretty amazing! No matter how the wheels are distributed across each quarter of the train car, we can get away with installing only a $O\left(\frac{\ln n}{n}\right)$ fraction of the track!

The algorithm and analysis described above can be summarized in the following theorem:

▶ **Theorem 1.1.** *Consider a $4f$-foot long train car, and suppose that the rear quarter of a train car contains n sets of wheels, each of which resides at a distinct integer distance from the rear of the car. In time $O(\ell n)$, one can construct an ℓ-foot long train track with the following two properties: (1) as the train car drives down the track at least one set of wheels from the rear of the car is always touching track; and (2) the track consists of 1-foot pillars, with the total expected number of pillars being at most $O\left(\frac{\ell \ln n}{n}\right)$.*

2 A Matching Lower Bound

In this section we show that using $O\left(\ell\frac{\ln n}{n}\right)$ pillars is optimal for some configurations of the train car. We are again going to use the probabilistic method, but this time in a more sophisticated way.

We continue to focus only on the rear quarter of the train car, which is f feet long. We set $f = 2n$, and we construct the rear quarter of the train car by placing n wheels at integer positions in the set $\{1, 2, \ldots, 2n\}$. We will then consider a track of length $\ell = 2f$, and show that $\Omega(\log n)$ pillars are necessary in order to support the rear quarter of the train car at all positions on the track. Recall that each pillar is one foot wide and is placed at an integer offset on the track.

Let C be the set of wheel-positions in the rear quarter of the train car. We choose C by placing a wheel at each position in $\{1, 2, \ldots, 2n\}$ independently with probability $\frac{1}{2}$. This means that C has n wheels in expectation, but may not actually have exactly n wheels. The important thing to note is that, with at least 50% probability, C has n or more wheels.

Now consider a track layout given by a subset T of $\{1, 2, \ldots, 4n\}$, and satisfying $|T| = \frac{\ln n}{4}$. Whereas C is a random variable, T is a fixed set.

Define $X_{C,T}$ to be the event that, for every possible position of the rear quarter of the train on the track, at least one wheel from the rear quarter of the train is supported by track. From the perspective of the train car, $X_{C,T}$ is a good event. Formally, $X_{C,T}$ occurs if for every offset $k \in \{0, 1, \ldots, 2n\}$, we have $(k + C) \cap T \neq \emptyset$.[1][2]

The key to proving the desired lower bound is to show that the probability of $X_{C,T}$ occurring is very small, namely that $\Pr[X_{C,T}] < \frac{1}{2\binom{4n}{(\ln n)/4}}$. Because T is a $\frac{\ln n}{4}$-element subset of $\{1, 2, \ldots, 4n\}$, there are at most $\binom{4n}{(\ln n)/4}$ possibilities for T. Taking a union bound over all of these possibilities implies that

$$\Pr[X_{C,T} \text{ for any } T] < \frac{1}{2}.$$

On the other hand, we know that the number of wheels $|C|$ is less than n with probability at most $1/2$. By a union bound, the probability that either $|C|$ has fewer than n wheels or that $X_{C,T}$ holds for some T is less than 1. It follows that there must exist a car C with n or more wheels for which no track T of size smaller than $(\ln n)/4$ satisfies $X_{C,T}$. In fact, with slightly more careful bookkeeping, one can show that an even stronger property is true: almost all choices of how to place n wheels in C require a track of size larger than $(\ln n)/4$ to support the car.

To complete the lower bound, the challenge becomes to show that $\Pr[X_{C,T}]$ is very small. That is, for a given choice of track T containing $(\ln n)/4$ pillars, the probability that T supports the rear-quarter of the train car C is small.

Rather than examine the event $X_{C,T}$ directly, we instead examine a related quantity. Define $Y_{C,T}$ to be the number of positions $k \in \{0, 1, \ldots, 2n\}$ for which $(k + C) \cap T = \emptyset$ (i.e., the number of positions in which the rear quarter of the car, given by C, falls through the track T).

The relationship between $X_{C,T}$ and $Y_{C,T}$ is that $X_{C,T}$ occurs only if $Y_{C,T} = 0$. Our approach to completing the analysis will be to first show that $\mathbb{E}[Y_{C,T}]$ is relatively large, and then to show that the probability of $Y_{C,T}$ deviating substantially from its expected value is small. This, in turn, implies that $\Pr[Y_{C,T} = 0]$ is small, completing the analysis. In other words, the problem of proving that there exists a train-car configuration requiring a large amount of track is reduced to the problem of proving a concentration inequality on the random variable $Y_{C,T}$.

For each position $k \in \{0, 1, \ldots, 2n\}$, the set $T - k$ consists of $(\ln n)/4$ elements. Since each of these elements is contained in C with probability $1/2$, the probability that C avoids all of the elements in $T - k$ is given by,

$$\frac{1}{2^{(\ln n)/4}} = \frac{1}{n^{1/4}}.$$

Summing over the values of k, it follows that the expected number of positions k in which $(C + k) \cap T = \emptyset$ is

$$\mathbb{E}[Y_{C,T}] = 2n \cdot \frac{1}{n^{1/4}} > n^{3/4}.$$

[1] Recall that the rear quarter of the train car is length $2n$, and that the train track is length $\ell = 4n$. We only wish to consider offsets k such that the rear quarter of the train car still sits entirely on potential track. That is, we wish to consider k such that $(k + \{1, 2, \ldots, 2n\}) \subseteq \{1, 2, \ldots, 4n\}$, meaning that the values of k that we care about are $k \in \{0, 1, \ldots, 2n\}$.

[2] For an integer r and a set S, we use $r + S$ to denote the set $\{s + r \mid s \in S\}$.

The final step of the analysis is to prove a concentration inequality on $Y_{C,T}$. Standard Chernoff bounds do not apply here because $Y_{C,T}$ is not a sum of independent indicator random variables. Instead, we employ a more powerful tool, namely McDiarmid's Inequality:

▶ **Theorem 2.1** (McDiarmid '89 [8]). *Let $A_1, \ldots, A_m$ be independent random variables over an arbitrary probability space. Let F be a function mapping $(A_1, \ldots, A_m)$ to $\mathbb{R}$, and suppose F satisfies,*

$$\sup_{a_1,a_2,\ldots,a_m,\overline{a_i}} |F(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m) - F(a_1, a_2, \ldots, a_{i-1}, \overline{a_i}, a_{i+1}, \ldots, a_m)| \leq R,$$

for all $1 \leq i \leq m$. That is, if $A_1, A_2, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m$ are fixed, then the value of A_i can affect the value of $F(A_1, \ldots, A_m)$ by at most R; this is known as the **Lipschitz Condition***. Then for all $S > 0$,*

$$\Pr[|F(A_1, \ldots, A_m) - \mathbb{E}[F(A_1, \ldots, A_m)]| \geq R \cdot S] \leq 2e^{-2S^2/m}.$$

To apply McDiarmid's Inequality to our situation, recall that $Y_{C,T}$ is defined to be the number of positions in the track T that the rear quarter of the car, given by C, falls though. Whereas the track T is fixed, each of the $2n$ possible wheels in C is picked with probability $1/2$. Define the indicator random variables $A_1, A_2, \ldots, A_{2n}$ so that A_i indicates whether $i \in C$. As required by McDiarmid's Inequality, the A_i's are independent of one-another, and $Y_{C,T}$ is a function of the A_i's.

Now we show that the Lipschitz condition holds with $R = (\ln n)/4$. Recall that the track T consists of only $(\ln n)/4$ pillars. Out of the $2n$ possible wheels i that C could contain, each of those wheels i is only relevant (to the car's stability) when the car is k feet down the track for some k that places wheel i on top of a pillar. Since there are only $(\ln n)/4$ pillars, each wheel i is only relevant to the train car's stability for $(\ln n)/4$ positions k on the track. In other words, for a given wheel position $i \in \{1, 2, \ldots, 2n\}$, there are only $(\ln n)/4$ values of $k \in \{0, 1, \ldots, 2n\}$ for which $(C + k) \cap T$ can possibly contain i. As a result, each A_i can only affect the value of $Y_{C,T}$ by at most $(\ln n)/4$, meaning that the Lipschitz condition holds with $R = (\ln n)/4$.

Applying McDiarmid's Inequality, we get that

$$\Pr[n^{3/4} - Y_{C,T} > n^{5/8} \cdot (\ln n)/4] \leq 2e^{-n^{1/4}}.$$

When n is large, this probability is much smaller than $\frac{1}{2\binom{4n}{(\ln n)/4}}$. It follows that $\Pr[X_{C,T}] = \Pr[Y_{C,T} = 0] < \frac{1}{2\binom{4n}{(\ln n)/4}}$. Summing over all possible options for the track T, the probability that any of them support the train car C is therefore less than $1/2$. It follows that some train car C with n or more wheels fails to be supported by any track T consisting of $(\ln n)/4$ or fewer pillars. This completes the lower bound, and establishes the following theorem.

▶ **Theorem 2.2.** *There exists a set of wheel positions $C \subseteq \{1, 2, \ldots, 2n\}$ such that $|C| \geq n$, and such that in order for a track $T \subseteq \{1, 2, \ldots, 4n\}$ to support the set of wheels at every position (meaning that $(C + k) \cap T \neq \emptyset$ for each $k \in \{0, \ldots, 2n\}$) the track T must have size $\Omega\left(\frac{\ln n}{n}\right)$.*

3 Three Algorithms for Building Track

In this section, we revisit the problem of constructing a train track that uses $O\left(\frac{\ell \ln n}{n}\right)$ feet of track, and present three alternative algorithms for this problem, each of which gives the same guarantees as the algorithm in Section 1.

We continue to assume that the wheels of the train car are at integer positions, and we focus only on the n wheels in the rear quarter of the train car. We use C to denote the set of positions of wheels, meaning that C is an n-element subset of $\{1, \ldots, f\}$. Our goal is to construct a set of pillars $T \subseteq \{1, 2, \ldots, \ell\}$ such that for each $k \in \{0, 1, \ldots, \ell - f\}$, the set $(C + k) \cap T$ is non-empty. As was the case in Section 1, we want an algorithm that runs in expected time $O(n\ell)$ and produces a set T with expected size $O\left(\frac{\ell \ln n}{n}\right)$.

Each of the three algorithms applies a different core technique from the overlap of probabilistic combinatorics and randomized algorithms:

- **A Deterministic Algorithm (Section 3.1).** The first algorithm uses the method of conditional probabilities to derandomize the algorithm given in Section 1.
- **An Application of the Algorithmic Lovász Local Lemma (Section 3.2)** The second algorithm uses the algorithmic version of the Lovász Local Lemma due to Moser and Tardos [9].
- **An Application of the Min-Hash Technique (Section 3.3)** The final algorithm uses a variant of the min-hash technique, which has previously found important applications in locality sensitive hashing and string alignment [2–4, 6, 7, 10].

3.1 A Deterministic Algorithm

In this section, we use the method of conditional probabilities [1] in order to design a deterministic algorithm for the train-track problem.

The basic idea behind the method of conditional probabilities is as follows. Suppose $X_1, \ldots, X_\ell$ are independent real-valued random variables, and that $F : \mathbb{R}^\ell \to \mathbb{R}$ is an objective function that we wish to minimize. We are given that $\mathbb{E}[F(X_1, \ldots, X_\ell)] \leq R$ for some value R, and we wish to find values of $x_1, \ldots, x_\ell \in \mathbb{R}$ for which $F(x_1, \ldots, x_\ell) \leq R$. The method of conditional probabilities takes an iterative approach. Suppose we already have values of $x_1, \ldots, x_k$ such that

$$\mathbb{E}[F(x_1, \ldots, x_k, X_{k+1}, \ldots, X_\ell)] \leq R.$$

Then there must exist some value x_{k+1} such that

$$\mathbb{E}[F(x_1, \ldots, x_k, x_{k+1}, X_{k+2}, \ldots, X_\ell)] \leq R. \tag{1}$$

The key challenge in applying the method of conditional probabilities is to design an objective function F that both captures the problem at hand, but that also allows for one to efficiently determine which value of x_{k+1} satisfies (1). This, in turn, allows for one to iteratively determine values for all of $x_1, \ldots, x_\ell$ such that $F(x_1, \ldots, x_\ell) \leq R$.

In order to apply the method of conditional probabilities to the train-track problem, we define $X_1, \ldots, X_\ell$ to be zero-one random variables, each of which takes value 1 with probability $\frac{\ln n}{n}$. Given values $x_1, \ldots, x_\ell$ for random variables $X_1, \ldots, X_\ell$, we can construct a train track T by first setting $T_1 = \{i \mid x_i = 1\}$, and then defining T to be T_1 with one additional pillar for each position k in which the train wheels C fall through the track T_1. Since our goal is to minimize the size of T, we define our objective function to be $F(x_1, \ldots, x_\ell) = |T|$.

In Section 1, we showed that $\mathbb{E}[F(X_1, \ldots, X_\ell)] \leq (1 + \ln n)/n$. Suppose that we have values $x_1, \ldots, x_k \in \{0, 1\}$ such that

$$\mathbb{E}[F(x_1, \ldots, x_k, X_{k+1}, \ldots, X_\ell)] \leq (1 + \ln n)/n. \tag{2}$$

Moreover, suppose that we maintain values $p_0, \dots, p_{\ell-f}$ such that each p_i denotes the probability that the set of wheels $(C+i)$ fall through the track $T = \{j \mid x_j = 1\} \cup \{j \mid X_j = 1\}$. This means that we can compute $\mathbb{E}[F(x_1, \dots, x_k, X_{k+1}, \dots, X_\ell)]$ as

$$\Big|\{i \mid x_i = 1\}\Big| + \frac{\ln n}{n}(\ell - k) + \sum_i p_i. \tag{3}$$

The first two terms represent $\mathbb{E}[|T_1|]$, and the third term represents $\mathbb{E}[|T \setminus T_1|]$.

Given values of $x_1, \dots, x_k$ such that (2) holds, we wish to find a value of $x_{k+1} \in \{0, 1\}$ so that (2) will hold for $k+1$. If we set $x_{k+1} = 1$, then this has the effect of increasing the expected initial size of $|T_1|$ by $1 - \frac{\ln n}{n}$, and of zeroing out any p_i's for which $k+1 \in (C+i)$. On the other hand, if we set $x_{k+1} = 0$, then this has the effect of decreasing the expected initial size of $|T_1|$ by $\frac{\ln n}{n}$ and replacing each p_i for which $k+1 \in (C+i)$ with $\frac{p_i}{1-(\ln n)/n}$. It follows that in time $O(n)$, one can update (3) in order to determine $\mathbb{E}[F(x_1, \dots, x_{k+1}, X_{k+2}, \dots, X_\ell)]$ for each of the two possible values of x_{k+1}. By selecting the value that minimizes the expected objective function, we can guarantee that

$$\mathbb{E}[F(x_1, \dots, x_{k+1}, X_{k+2}, \dots, X_\ell)] \le \frac{1 + \ln n}{n}.$$

Continuing like this, we can find values of $x_1, \dots, x_\ell$ such that $F(x_1, \dots, x_\ell) \le (1 + \ln n)/n$ in time $O(\ell n)$. Using these $x_1, \dots, x_\ell$ to construct the track T results in a track that uses at most $(1 + \ln n)/n$ pillars, as desired.

3.2 An Application of the Algorithmic Lovász Local Lemma

Given a large collection of unlikely events $E_1, \dots, E_m$, such that each event E_i is related to only a small number of other events E_j, the Lovász Local Lemma is a technique for showing that there exists a way for all m events to mutually not occur. In one of its most basic forms, the Lovász Local Lemma can be stated as follows:

▶ **Theorem 3.1** (Lovász and Erdös '73 [5]). *Suppose $X_1, \dots, X_s$ are independent random variables, possibly over different probability spaces. Let $E_1, \dots, E_m$ be events such that each E_i is determined by some subset of the X_j's – that is, there exists an index set $I_i \subseteq [s]$ such that E_i is determined by the outcome of the X_j's for which $j \in I_i$. Say that two events E_i and E_j* **depend on each other** *if $I_i \cap I_j \neq \emptyset$. Let p be such that $\Pr[E_i] \le p$ for each i, and let d be such that each E_i depends on at most d different E_j's (including E_i). If $pde \le 1$, where e is the universal constant, then there is a positive probability that none of the events $E_1, \dots, E_m$ occur.*

The algorithmic version of the Lovász Local Lemma gives an efficient algorithm for constructing values of $X_1, \dots, X_s$ in order to ensure that none of the events $E_1, \dots, E_m$ occur.

▶ **Theorem 3.2** (Moser and Tardos '10 [9]). *Suppose that the conditions from Theorem 3.1 hold. Consider the following algorithm for choosing values of $X_1, \dots, X_s$: First independently sample each of $X_1, \dots, X_s$ from its defining probability distribution; then, as long as their exists at least one event E_i that holds, pick such an event E_i and resample the X_j's for each $j \in I_i$. Each time that the X_j's are resampled for some event E_i, we call the resamplings a* **phase** *of the algorithm. The algorithm terminates once the X_i's have been assigned values that result in no events E_i occurring.*

The above algorithm, known as the **fix-it algorithm**, *terminates in finite expected time, and the expected number of phases is at most n/d.*

In order to apply the Algorithmic Lovász Local Lemma to our problem, we define $X_1, \ldots, X_\ell$ to be independent zero-one random variables, each taking value 1 with probability $\frac{1+2\ln n}{n}$. Each X_i is the indicator variable for whether we include pillar i in the train track. We then define events $E_0, \ldots, E_{\ell-f}$ so that E_i is the event that the rear-quarter of the train car falls through the track at position i. That is, E_i occurs if $(C+i) \cap \{j \mid X_j = 1\} = \emptyset$.

Each event E_i depends on only n different X_j's, and each X_j is relevant to only n different E_k's. It follows that each event E_i depends on at most n^2 other E_k's (including E_i). This means that we can apply the Algorithmic Lovász Local Lemma with $d = n^2$.

In order for a given event E_i to occur, there are n different pillars that all must fail to appear in the track. The probability of this happening is

$$\Pr[E_i] = \left(1 - \frac{1+2\ln n}{n}\right)^n < \frac{1}{e^{1+2\ln n}} \leq \frac{1}{en^2}.$$

Using $d = n^2$ an $p = \frac{1}{en^2}$, we can apply the Lovász Local Lemma in order to conclude that there exists a choice of pillars $X_1, \ldots, X_\ell$ so that the wheels in C are supported along the entire track. This alone is not a useful observation since, of course, setting $X_1, \ldots, X_\ell$ all to 1 would trivially support the wheels in C at all points. On the other hand, if we apply the algorithmic version of the Lovász Local Lemma, then get an additional fact: that the fix-it algorithm terminates after only $O(\ell/n^2)$ phases in expectation.

Since each phase resamples only n different X_i's, the resamples contribute at most $O(\ell/n)$ pillars in expectation. On the other hand, the initial configuration of the X_i's contributes at most $(1+2\ln n)/n$ pillars in expectation. It follows that, at the end of the fix-it algorithm, the resulting track configuration will use at most $(2+2\ln n)/n$ pillars in expectation. A careful implementation of the fix-it algorithm will run in expected time $O(\ell n)$, as desired.

3.3 An Application of Min-Hash

Given a collection of sets $\mathcal{S}$, *Min-Hashing* is a technique for randomly sampling one element for each set $S \in \mathcal{S}$. The technique works by first hashing each element s of each set S in $\mathcal{S}$ to a random real number $h(s) \in (0,1)$. For each set $S \in \mathcal{S}$, one then samples the element $s \in S$ with minimum hash $h(s)$.

The Min-Hashing technique plays important roles in both Locality Sensitive Hashing [2,3,7] and string-alignment algorithms [4,6,10]. The key property of Min-Hashing is that if two sets $S_1, S_2 \in \mathcal{S}$ are similar to one-another, then their min-hash is likely to be the same. And more generally, if an element s is the minimum-hashed element in one set $S \in \mathcal{S}$, then s is likely to also be the minimum-hashed element in other sets.

In our application of Min-Hashing, we need not actually use hash functions. Instead, we assign random real numbers $r_1, \ldots, r_\ell \in (0,1)$ to each of the ℓ possible track pillars. For each possible offset $k \in \{0, 1, 2, \ldots, \ell - f\}$, define the set $S_k = (C+k)$ to be the positions that the wheels in C take when the train car is k feet down the track. We construct a train track T by adding the pillar $\operatorname{argmin}_{s \in S_k} r_s$ for each set S_k. That is, for each position that the train could sit in the track, we look at all possible pillars that could hold the rear-quarter of the train up, and we include in our track the pillar with the minimum assigned random value r_s. We say that this pillar s is ***sampled*** from S_k.

By construction, the set of pillars T is guaranteed to support the wheels C at every position. What is less clear is whether $|T|$ will be small. Here is where we take advantage of the properties of Min-Hashing, and the fact that many of the sets S_k sample the same pillars as one another.

The key observation is that almost all of the pillars s that are sampled have small random values r_s. Consider, in particular, the probability that for a given set S_k, we sample a pillar s for which $r_s > (\ln n)/n$. This means that all n pillars in S_k were assigned random values larger than $(\ln n)/n$, which happens with probability at most,

$$\left(1 - \frac{\ln n}{n}\right)^n \leq \frac{1}{e^{\ln n}} = \frac{1}{n}.$$

It follows that, out of the $\ell - f$ samplings that occur, the expected number of pillars s for which $r_s > (\ln n)/n$ that are sampled is at most $(\ell - f)/n \leq \ell/n$. On the other hand, even if every pillar s for which $r_s \leq (\ln n)/n$ is sampled, the expected number of them is at most $\ell(\ln n)/n$. The total number of sampled pillars, and thus the size of T, is therefore at most $\ell(1 + \ln n)/n$, in expectation. This completes the analysis of the algorithm.

References

1 Noga Alon and Joel H Spencer. *The probabilistic method.* John Wiley & Sons, 2004.

2 Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of Sequences 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

3 Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

4 Moses Charikar, Ofir Geri, Michael P Kim, and William Kuszmaul. On estimating edit distance: Alignment, dimension reduction, and embeddings. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

5 Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Colloquia Mathematics Societatis Janos Bolai 10. Infinite and Finite Sets, Keszthely (Hungary)*. Citeseer, 1973.

6 William Kuszmaul. Efficiently approximating edit distance between pseudorandom strings. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1165–1180. Society for Industrial and Applied Mathematics, 2019.

7 Mark S Manasse. On the efficient determination of most near neighbors: horseshoes, hand grenades, web search and other situations when close is close enough. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 4(4):1–88, 2012.

8 Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

9 Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

10 Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):132, 2016.

Card-Based ZKP Protocols for Takuzu and Juosan

Daiki Miyahara
Graduate School of Information Sciences,
Tohoku University, Sendai, Japan
National Institute of Advanced Industrial Science
and Technology (AIST), Tokyo, Japan
daiki.miyahara.q4@dc.tohoku.ac.jp

Léo Robert
University Clermont Auvergne, LIMOS,
CNRS UMR (6158), Aubière, France
leo.robert@uca.fr

Pascal Lafourcade
University Clermont Auvergne, LIMOS,
CNRS UMR (6158), Aubière, France
pascal.lafourcade@uca.fr

So Takeshige
School of Engineering, Tohoku University,
Sendai, Japan
so.takeshige.q1@dc.tohoku.ac.jp

Takaaki Mizuki
Cyberscience Center, Tohoku University,
Sendai, Japan
tm-paper+zerotate@g-mail.tohoku-university.jp

Kazumasa Shinagawa
Graduate School of Informatics and Engineering, The University of Electro-Communications,
Tokyo, Japan
National Institute of Advanced Industrial Science
and Technology (AIST), Tokyo, Japan
shinagawakazumasa@uec.ac.jp

Atsuki Nagao
Department of Information Science,
Ochanomizu University, Tokyo, Japan
a-nagao@is.ocha.ac.jp

Hideaki Sone
Cyberscience Center, Tohoku University,
Sendai, Japan

Abstract

Takuzu and Juosan are logical Nikoli games in the spirit of Sudoku. In Takuzu, a grid must be filled with 0's and 1's under specific constraints. In Juosan, the grid must be filled with vertical and horizontal dashes with specific constraints. We give physical algorithms using cards to realize zero-knowledge proofs for those games. The goal is to allow a player to show that he/she has the solution without revealing it. Previous work on Takuzu showed a protocol with multiple instances needed. We propose two improvements: only one instance needed and a soundness proof. We also propose a similar proof for Juosan game.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques

Keywords and phrases Zero-knowledge proof, Card-based cryptography, Takuzu, Juosan

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.20

Funding *Daiki Miyahara*: This work was supported by JSPS KAKENHI Grant Number JP19J21153.
Léo Robert: This work was partially supported by the French project ANR-18-CE39-0019 (MobiS5).
Pascal Lafourcade: This work was partially supported by the project ANR-18-CE39-0019 (MobiS5).
Takaaki Mizuki: This work was supported by JSPS KAKENHI Grant Number JP17K00001.
Kazumasa Shinagawa: This work was supported by JSPS KAKENHI Grant Number JP17J01169.

Acknowledgements We thank the anonymous referees, whose comments have helped us to improve the presentation of the paper. In particular, Protocol 1 for Takuzu presented in Section 3.2.1 is based on the fruitful comments given by one referee.

1 Introduction

James Bond and Q decide to spend most of their holidays on the Spiaggia Praia beach (located at Isola di Favignana, Sicily, Italy). Before swimming in the sea, they like to play

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 20; pp. 20:1–20:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with logical games. James Bond is a specialist of *Takuzu*. Takuzu is a puzzle invented by Frank Coussement and Peter De Schepper in 2009[1]. It was also called *Binero*, *Bineiro*, *Binary Puzzle*, *Brain Snacks* or *Zernero*. Figure 1 contains a simple Takuzu grid and its solution. Q is an expert of *Juosan*, which was published by Nikoli[2]. Figure 2 contains a Juosan grid and its solution.

Each one proposes his favorite game to the other as a challenge. Both are competitive, and each challenge ends to be so hard that the other cannot solve it. James Bond immediately supposes that something is wrong and asks Q a proof that the grid has a solution. Of course, Q thinks the same way about Bond's challenge. Since they are both suspicious, they want to prove that there is a solution without giving any information about the solution.

In cryptography, the process, which allows a party to prove that it has a data without leaking any information on this data, is called Zero-Knowledge Proof (ZKP).

More formally, a ZKP is a protocol which enables a prover P to convince that it has a solution s of a problem to a verifier V. This proof cannot leak any information on s. The protocol must observe three properties.

- **Completeness:** If P knows s then it can convince V.
- **Soundness:** If P does not know s, it can convince V with only a negligible probability.
- **Zero-Knowledge:** V learns nothing about s. This can be formalized by showing that the outputs of a simulator and outputs of the real protocol follow the same probability distribution.

The concept of interactive ZKP was introduced by Goldwasser et al. [12]. Then it was shown that for any NP complete problem there exists an interactive ZKP protocol [11]. There is also an extension showing that every provable statement can be proved in zero-knowledge [2].

There exist protocols where the prover and the verifier do not need to interact. Such protocols are called non-interactive ZKP [4]. For a complete background on ZKP's, see [18].

Usually ZKP protocols are executed by computers, yet, our aim is to design a solution for Bond and Q's dilemma using physical objects such as cards, since on the Spiaggia Praia beach they do not want to use their computers. We first recall the rules of these two games before presenting our contributions.

Takuzu's Rules

The goal of Takuzu is to fill a rectangular grid of even size with 0's and 1's. An initial Takuzu grid already contains a few filled cases. A grid is solved when it is full (*i.e.*, no empty cases) and respects the following constraints.

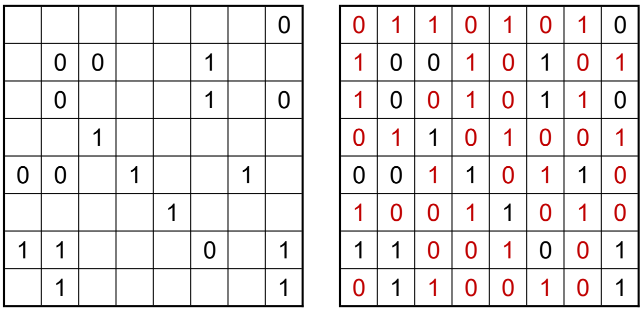
1. **Equality Rule:** Each row/column contains exactly the same number of 1's and 0's.
2. **Uniqueness Rule:** Each row (column) is unique among all rows (columns).
3. **Adjacent Rule:** In each row and each column there can be no more than two same numbers adjacent to each other; for example 110010 is possible, but 110001 is impossible.

The problem of solving a Takuzu grid was proven to be NP complete in [3, 34].

Juosan's Rules

A Juosan grid is divided into territories by bold lines, where a territory is possibly associated with a number. The goal is to fill in all cells with a vertical (|) or horizontal (–) dash such

[1] https://en.wikipedia.org/wiki/Takuzu
[2] http://www.nikoli.co.jp/en/puzzles/juosan.html



							0
	0	0			1		
	0				1		0
		1					
0	0		1			1	
				1			
1	1				0		1
	1						1

0	1	1	0	1	0	1	0
1	0	0	1	0	1	0	1
1	0	0	1	0	1	1	0
0	1	1	0	1	0	0	1
0	0	1	1	0	1	1	0
1	0	0	1	1	0	1	0
1	1	0	0	1	0	0	1
0	1	1	0	0	1	0	1

Figure 1 Example of a 8 × 8 Takuzu challenge, and its solution. We can verify that each row and column is unique, contains the same number of 0's and 1's, and there are never three consecutive 1's or 0's.

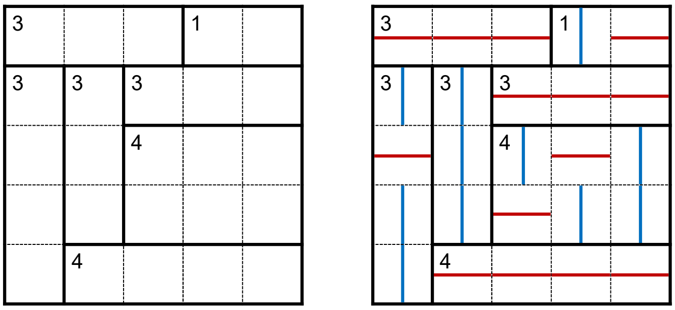


Figure 2 Example of a Juosan challenge, and its solution from Nikoli website.

that the following three constraints are satisfied.

1. **Room Rule:** The number in every territory equals the number of either vertical or horizontal dashes in it (in some cases, there may be equal numbers of both). Territories with no number may have any number of vertical dashes and horizontal dashes.
2. **Adjacent (horizontal) Rule:** Horizontal dashes can extend more than three cells horizontally but no more than two cells vertically.
3. **Adjacent (vertical) Rule:** Vertical dashes can extend more than three cells vertically but no more than two cells horizontally.

In 2018, the problem of solving a Juosan grid was proven to be NP complete in [16].

Contributions

We have the two main following contributions.

1. We propose better ZKP protocols for Takuzu which improve upon the approach given in [5]. The latter used several instances of the protocol while ours use only one instance. We also improve the soundness of the proof in the sense that if the prover does not have a solution, he convinces the verifier with null probability.
2. We also propose an adapted version of this technique to Juosan. Again, only one instance of the protocol is run for proving to V that if P does not know the solution, then P convinces V with probability 0. We also propose an optimized version of the Adjacent Verification which aims to show validity of four consecutive commitments.

Related Work

There are works on implementing cryptographic protocols using physical objects, as in [23] for example, or in [9] where a physical secure auction protocol was proposed. Other implementations have been studied using cards in [8], polarizing plates [32], polygon cards [31], a standard deck of playing cards [20], using a PEZ dispenser [1], using a dial lock [21], using a 15 puzzle [22], or using a tamper-evident seals [25, 26, 27].

In FUN'18, the authors of [29] revisited the ZKP for Sudoku proposed by Gradwohl et al. in FUN'07 [13]. This is a clear progress in the construction of ZKP since the technique proposed in this paper uses specific protocols to perform zero-knowledge proof for Sudoku. Indeed, those protocols use a normal deck of playing cards and have no soundness error with a reasonable number of playing cards. The original technique for Sudoku was extended for Hanje [7]. ZKP's for several other puzzles have been studied such as Akari [5], Takuzu [5], Kakuro [5, 19], KenKen [5], Makaro [6], Norinori [10], and Slitherlink [17].

There is a ZKP proof for Takuzu puzzle [5] (recall in Section 2), but we propose an enhanced version using only one instance of the protocol to convince the verifier. The previous proof is decomposed into several cases to avoid leak of information toward the solution. This implies the need of rerunning the protocol several times for completely convincing V that P has the solution. The construction of the protocol leads to have a negligible probability that the prover P does not know the solution. Our proof is designed in such a way that only one instance is run leading to a complete soundness of the proof (i.e., if P does not have the solution, the probability of convincing V is null). We show that this technique can be adapted to Juosan game which has not been studied before. The detailed security proof for our ZKP protocols for Takuzu is given in Section 3.4 and for Juosan in Section 4.4.

Outline: In Section 2, we present an existing ZKP protocol for Takuzu. In Section 3, we improve the ZKP protocol for Takuzu. In Section 4, we present our ZKP protocol for Juosan. In the last section we conclude.

2 The Existing ZKP Protocol for Takuzu

We give a ZKP proof using physical objects. The goal is to show that the prover P (aka James Bond) can prove to the verifier V (aka Q) that he knows a solution of a given Takuzu grid. The material used for the proof include two printed grids on a sheet of paper, a piece of paper, an envelope and two kinds of cards: cards with a 0 or a 1 printed on them.

There are two phases in this protocol, the Setup which generates the permutations used for the second phase called the verification.

Let G be the $n \times n$ initial Takuzu grid and S the matrix relative to the solution known by P (including the initial cells).

Setup. The prover P chooses uniformly at random two permutations: π_R for the rows, and π_C for the columns. He writes the two permutations on a paper and place the latter into an envelope E. Then he computes $S' = \pi_R(\pi_C(S))$. Finally, P places cards face down on the second grid according to S'. We denote the configuration of these cards by the matrix $\tilde{S}'$

Verification. The verifier V picks c randomly among $\{0, 1, 2, 3\}$.

If $c = 0$: This case corresponds to P proving that the solution is the one of the initial grid. V computes $G' = \pi_R(\pi_C(G))$ with the permutations found in the envelope E. Then V determines the cells of G' corresponding to the initial cells of G. Finally, V checks if the revealed cards are the same as the one revealed in the second grid (that are placed according to $\tilde{S}'$).

If $c = 1$: This case corresponds to P proving that adjacent rule holds.

V permutes (face down) the cards of $\tilde{S}'$ to obtain $\tilde{S} = \pi_c^{-1}(\pi_R^{-1}(\tilde{S}'))$ using the permutations in E. Then, V picks d randomly among $\{0, 1\}$ and e randomly among $\{1, 2, 3\}$.

If $d = 0$: For each row, V sets $x = \lfloor \frac{n-e}{3} \rfloor$ decks of three cards $\{(e + 3 \cdot i + 1, e + 3 \cdot i + 2, e + 3 \cdot i + 3)\}_{\{0 \leq i < x\}}$ where the triplet (i, j, k) denotes a deck containing the i^{th}, the j^{th} and the k^{th} cards of the row.

If $d = 1$: For each column, V sets $x = \lfloor \frac{n-e}{3} \rfloor$ decks of three cards $\{(e + 3 \cdot i + 1, e + 3 \cdot i + 2, e + 3 \cdot i + 3)\}_{\{0 \leq i < x\}}$ where the triplet (i, j, k) denotes a deck containing the i^{th}, the j^{th} and the k^{th} cards of the column.

Then, V gives the triplets to P. For each deck, P removes one of the two identical cards. Then P reveals the cards to V, who accepts only if he sees two different cards.

If $c = 2$: This case corresponds to P proving that uniqueness rule holds.

For this, V picks randomly one row or one column. V reveals all the cards of his chosen row (or column). For each of the $n - 1$ other rows (or columns) the verifier picks the cards where a 0 appears in the revealed rows (or column). At this step, V does not reveal those cards. Each one of these $n - 1$ sets of cards is shuffled by the shuffle functionality and given back to the prover. P reveals one card per set that is a 1. Thus each one of the other $n - 1$ rows (or columns) are different from the revealed row, since the initial row (or column) has a 0 where the other column (or row) has a 1. If there are several 1's in a deck, the prover randomly chooses which one to reveal.

If $c = 3$: This case corresponds to P proving that the equality rule holds.

The verifier V picks d randomly among $\{0, 1\}$.

If $d = 0$, for each row, V takes all the cards in the row and keep them face down. Then V gathers the cards in order to shuffle those n decks. We assume that the verifier has access to a *shuffle functionality* which is essentially an indistinguishable shuffle of face down cards. Note that this action could be done by a trusted third party (M for instance) but not by P or V (since they could cheat and modify the cards).

Finally, V checks that each deck contains exactly the same number of 1's and 0's.

If $d = 1$, the same process is done except that V picks columns instead of rows.

To have the best security guarantees, the verifier should choose his challenges c, d, etc. such that each combination of challenges at the end has the same probability. This protocol is repeated k times where k is a chosen security parameter. Note that the ZKP is again polynomial in the size of the grid.

3 Our improved ZKP Protocols for Takuzu

In this section, we propose two ZKP protocols for Takuzu; our protocols are simple and have no soundness error. Remember that the goal is to show the prover P (aka James Bond) can prove to the verifier V (aka Q) that P knows a solution of a given Takuzu grid.

Our protocols use black cards $\boxed{\clubsuit}$, red cards $\boxed{\heartsuit}$, and number cards $\boxed{1}\,\boxed{2}\cdots\boxed{6}$ whose backs $\boxed{?}$ are all identical. In the sequel, we use the following encoding rule:

$$\boxed{\clubsuit}\boxed{\heartsuit} = 0, \quad \boxed{\heartsuit}\boxed{\clubsuit} = 1. \tag{1}$$

That is, black-to-red represents 0 and red-to-black represents 1. We call two face-down cards that correspond to a bit $x \in \{0, 1\}$ according to the above encoding rule (1) a *commitment to x*, and we write it as $\underbrace{\boxed{?}\boxed{?}}_{x}$. Roughly, our improved ZKP protocols for Takuzu proceed as follows.

Table 1 The exact values of $|\mathsf{tkz}(n)|$ when n is up to ten.

n	$\|\mathsf{tkz}(n)\|$
4	6
6	14
8	34
10	84

Setup phase: The prover P places a commitment to each cell according to the solution.

Verification phases: The verifier V verifies that the placement of the commitments satisfies all the constraints.

To present the complete description of our protocols in Section 3.2, we show some preliminaries in Section 3.1. In Section 3.3, we show that there is a trade-off between our two protocols and compare them.

3.1 Preliminaries

In this subsection, we introduce some notations and two subprotocols, which will be used to present our constructions in Section 3.2.

3.1.1 Possible Sequences

For an even number n, we denote by $\mathsf{tkz}(n)$ the set of all binary sequences satisfying the Uniqueness and Equality rules of Takuzu, that is, $\mathsf{tkz}(n) := \{w \in \{0,1\}^n \mid w$ contains exactly $n/2$ 0's and no three consecutive digits$\}$. For example, $\mathsf{tkz}(4) = \{0011, 1100, 0101, 1010, 0110, 1001\}$. The size of $\mathsf{tkz}(n)$ can be computed as Table 1. The size $|\mathsf{tkz}(n)|$ is known in the On-line Encyclopedia of Integer Sequences (OIES) as "the number of paths from $(0,0)$ to (n,n) avoiding 3 or more consecutive east steps and 3 or more consecutive north steps.[3]" We can also show that $\mathsf{tkz}(n) = O((\frac{3+\sqrt{5}}{2})^n n^{-\frac{1}{2}})$.

3.1.2 Basic Shuffles

Pile-scramble shuffle [15]. This is the following shuffling operation: Given a sequence of m piles, each of which consists of the same number of face-down cards, denoted by $\underbrace{\boxed{?}}_{\boldsymbol{p}_1}\,\underbrace{\boxed{?}}_{\boldsymbol{p}_2}\cdots\underbrace{\boxed{?}}_{\boldsymbol{p}_m}$, applying a *pile-scramble shuffle* (denoted by $[\cdot|\ldots|\cdot]$) results in $\left[\,\underbrace{\boxed{?}}_{\boldsymbol{p}_1}\,\middle|\,\underbrace{\boxed{?}}_{\boldsymbol{p}_2}\,\middle|\cdots\middle|\,\underbrace{\boxed{?}}_{\boldsymbol{p}_m}\,\right] \rightarrow \underbrace{\boxed{?}}_{\boldsymbol{p}_{r^{-1}(1)}}\,\underbrace{\boxed{?}}_{\boldsymbol{p}_{r^{-1}(2)}}\cdots\underbrace{\boxed{?}}_{\boldsymbol{p}_{r^{-1}(m)}}$, where $r \in S_m$ is a uniformly distributed random permutation and S_m denotes the symmetric group of degree m. To implement a pile-scramble shuffle, we use physical cases that can store a pile of cards, such as boxes and envelopes; a player (or players) randomly shuffle them until nobody traces the order of the piles.

[3] https://oeis.org/A177790

Pile-shifting shuffle. A *pile-shifting shuffle* (or a pile-shifting scramble [28]) is to *cyclically* shuffle piles of cards. That is, given m piles, applying a pile-shifting shuffle (denoted by $\langle \cdot|\ldots|\cdot\rangle$) results in $\Big\langle \underbrace{\boxed{?}}_{\boldsymbol{p}_1} \Big| \underbrace{\boxed{?}}_{\boldsymbol{p}_2} \Big| \cdots \Big| \underbrace{\boxed{?}}_{\boldsymbol{p}_m} \Big\rangle \rightarrow \underbrace{\boxed{?}}_{\boldsymbol{p}_{s+1}} \underbrace{\boxed{?}}_{\boldsymbol{p}_{s+2}} \cdots \underbrace{\boxed{?}}_{\boldsymbol{p}_{s+m}}$, where s is uniformly and randomly chosen from $\mathbb{Z}/m\mathbb{Z}$. To implement a pile-shifting shuffle, we use similar materials as a pile-scramble shuffle; a player (or players) cyclically shuffle them by hand until nobody traces the offset.

3.1.3 Mizuki–Sone AND (OR) Protocol

Given two commitments to $a, b \in \{0,1\}$ (along with additional two cards $\boxed{\clubsuit}\boxed{\heartsuit}$), the Mizuki–Sone AND protocol [24] outputs a commitment to $a \wedge b$: $\underbrace{\boxed{?}\boxed{?}}_{a}\ \underbrace{\boxed{?}\boxed{?}}_{b}\ \boxed{\clubsuit}\boxed{\heartsuit} \rightarrow \cdots \rightarrow \underbrace{\boxed{?}\boxed{?}}_{a\wedge b}$.

Note that the output commitment can be used for another protocol. The protocol proceeds as follows.

1. Rearrange the sequence as follows: $\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}}\overset{6}{\boxed{?}} \rightarrow \overset{1}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{2}{\boxed{?}}\overset{5}{\boxed{?}}\overset{6}{\boxed{?}}$.
2. Apply a *random bisection cut*: $[\,\boxed{?}\boxed{?}\boxed{?}\,|\,\boxed{?}\boxed{?}\boxed{?}\,] \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$. A random bisection cut is a special case of a pile-scramble shuffle; it bisects a sequence of cards and then shuffles the two halves.
3. Reveal the first and fourth cards in the sequence. Then, the output commitment can be obtained as follows: $\boxed{\clubsuit}\boxed{?}\boxed{?}\boxed{\heartsuit}\underbrace{\boxed{?}\boxed{?}}_{a\wedge b}$ or $\boxed{\heartsuit}\underbrace{\boxed{?}\boxed{?}}_{a\wedge b}\boxed{\clubsuit}\boxed{?}\boxed{?}$.

Note that by De Morgan's laws we can have the Mizuki–Sone OR protocol that produces a commitment to $a \vee b$ given two commitments to a and b.

3.1.4 Mizuki–Sone XOR protocol

Given two commitments to $a, b \in \{0,1\}$, the Mizuki–Sone XOR protocol [24] outputs a commitment to $a \oplus b$: $\underbrace{\boxed{?}\boxed{?}}_{a}\ \underbrace{\boxed{?}\boxed{?}}_{b} \rightarrow \cdots \rightarrow \underbrace{\boxed{?}\boxed{?}}_{a\oplus b}$. The protocol proceeds as follows.

1. Rearrange the sequence as follows: $\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}} \rightarrow \overset{1}{\boxed{?}}\overset{3}{\boxed{?}}\overset{2}{\boxed{?}}\overset{4}{\boxed{?}}$.
2. Apply a random bisection cut to the sequence: $[\,\boxed{?}\boxed{?}\,|\,\boxed{?}\boxed{?}\,] \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}$.
3. Rearrange the sequence as follows: $\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}} \rightarrow \overset{1}{\boxed{?}}\overset{3}{\boxed{?}}\overset{2}{\boxed{?}}\overset{4}{\boxed{?}}$.
4. Reveal the first and second cards in the sequence. Then, the output commitment can be obtained as follows: $\boxed{\clubsuit}\boxed{\heartsuit}\underbrace{\boxed{?}\boxed{?}}_{a\oplus b}$ or $\boxed{\heartsuit}\boxed{\clubsuit}\underbrace{\boxed{?}\boxed{?}}_{\overline{a\oplus b}}$.

3.1.5 Six-Card Trick

Given three commitments to $a, b, c \in \{0,1\}$, the *six-card trick* [30][4] outputs 1 if $a = b = c$ and 0 otherwise: $\underbrace{\boxed{?}\boxed{?}}_{a}\ \underbrace{\boxed{?}\boxed{?}}_{b}\ \underbrace{\boxed{?}\boxed{?}}_{c} \rightarrow \cdots \rightarrow \begin{cases} 1 & \text{if } a = b = c, \\ 0 & \text{otherwise.} \end{cases}$

That is, we can know only whether the values of given three commitments are the same or not by using the six-card trick. We use it in our construction to verify the Adjacent rule.

[4] The protocol had been invented independently by Heather, Schneider, and Teague [14].

The protocol proceeds as follows.

1. Rearrange the sequences as follows: $\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}}\overset{6}{\boxed{?}} \to \overset{1}{\boxed{?}}\overset{6}{\boxed{?}}\overset{3}{\boxed{?}}\overset{2}{\boxed{?}}\overset{5}{\boxed{?}}\overset{4}{\boxed{?}}$.
2. Apply a *random cut* (which is denoted by $\langle\cdots\rangle$) to the sequence: $\left\langle \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \right\rangle \to \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$. A random cut is a special case of a pile-shifting shuffle; it cyclically shuffles a sequence of cards. Note that a random cut can be implemented easily with human hands [33].
3. Reveal the sequence.
   a. If the resulting sequence is $\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}$ (apart from cyclic shifts), the output is 1, i.e., $a = b = c$ holds.
   b. If the resulting sequence is $\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}$ (apart from cyclic shifts), the output is 0, i.e., $a = b = c$ does not hold.

3.1.6 Input-Preserving Function Evaluation Technique

As seen in Section 3.1.5, we can know whether the equality of three input commitments holds although the input commitments are destroyed after executing the six-card trick. The *input-preserving function evaluation technique* enables us to obtain input commitments again after some function evaluation (such as the equality) by using some number cards.

Let us first explain the *input-preserving six-card trick* as follows.

1. Place a number card below each card, and then turn them over:

$$\underbrace{\boxed{?}\boxed{?}}_{a}\ \underbrace{\boxed{?}\boxed{?}}_{b}\ \underbrace{\boxed{?}\boxed{?}}_{c} \to \begin{array}{c}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\\ \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}\end{array} \to \begin{array}{c}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\\ \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\end{array}.$$

2. Rearrange the sequences as follow: $\begin{array}{c}\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}}\overset{6}{\boxed{?}}\\ \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\end{array} \to \begin{array}{c}\overset{1}{\boxed{?}}\overset{6}{\boxed{?}}\overset{3}{\boxed{?}}\overset{2}{\boxed{?}}\overset{5}{\boxed{?}}\overset{4}{\boxed{?}}\\ \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\end{array}$.
3. Apply a pile-shifting shuffle to the sequences:

$$\left\langle \begin{array}{c|c|c|c|c|c}\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}\\ \boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}\end{array} \right\rangle \to \begin{array}{c}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\\ \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\end{array}.$$

4. Reveal the cards of all sequences except for the number cards; then, we obtain the output as shown in Step 3 in Section 3.1.5.
5. Turn over the face-up cards and apply a pile-scramble shuffle to the sequences:

$$\left[\begin{array}{c|c|c|c|c|c}\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}\\ \boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}&\boxed{?}\end{array} \right] \to \begin{array}{c}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\\ \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\end{array}.$$

6. Reveal the number cards and rearrange the sequence of piles so that the revealed number cards become in ascending order; then, we have restored input commitments to a, b, and c. The following is an example case:

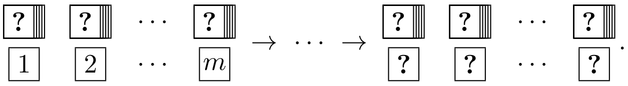
$$\begin{array}{c}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\\ \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\end{array} \to \begin{array}{c}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\\ \boxed{3}\boxed{1}\boxed{5}\boxed{4}\boxed{6}\boxed{2}\end{array} \to \begin{array}{c}\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{b}\underbrace{\boxed{?}\boxed{?}}_{c}\\ \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}\end{array}.$$

More formally, assume that we have a protocol to evaluate some function with m input piles of cards. Then, the input-preserving function evaluation technique enables us to obtain m input piles again after some function evaluation by using m number cards:

$$\begin{array}{ccc}\boxed{?}\!\Vert & \boxed{?}\!\Vert & \cdots & \boxed{?}\!\Vert\\ \boxed{1} & \boxed{2} & \cdots & \boxed{m}\end{array} \to \cdots \to \text{some function evaluation} \to \cdots \to \boxed{?}\!\Vert\,\boxed{?}\!\Vert \cdots \boxed{?}\!\Vert.$$

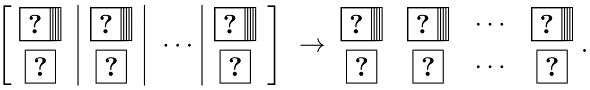
This proceeds as follows.

1. Attach a corresponding number card to each of m input piles:



Together with the added number cards, execute a designated protocol to evaluate some function.

2. Apply a pile-scramble shuffle to the sequence of piles:



3. Reveal only the number cards. Then, rearrange the sequence of piles so that the revealed number cards become in ascending order to obtain m input piles.

3.2 Our Constructions

We are now ready to present the full description of our ZKP protocols for Takuzu, namely Protocols 1 and 2.

3.2.1 Protocol 1: Verifying Each Constraint Separately

Given a Takuzu puzzle instance of $n \times n$ grid, *Protocol 1* verifies that all the constraints, namely the Equality, Uniqueness, and Adjacent rules, are satisfied separately.

Setup phase. Remember the encoding rule (1). The prover P places a commitment on each cell according to the solution (which is kind of a (0,1)-matrix).

Adjacent Verification phase. In this phase, V verifies that the Adjacent rule is satisfied. For this, V repeats the following for every three consecutive commitments in rows and columns.

1. Attach the corresponding number card to each of the six cards:

$$\boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\,\boxed{?}\boxed{?} \rightarrow \begin{array}{c}\boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\\ \boxed{1}\boxed{2}\,\boxed{3}\boxed{4}\,\boxed{5}\boxed{6}\end{array} \rightarrow \begin{array}{c}\boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\\ \boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\,\boxed{?}\boxed{?}\end{array}.$$

2. Perform the input-preserving six-card trick shown in Section 3.1.6 to prove that the three commitments are not all 0s and 1s. If the six-card trick outputs 1, V rejects it.

Uniqueness Verification phase. In this phase, V verifies that the Uniqueness rule is satisfied. V repeats the following for every pair of rows (and columns), each of which consists of n commitments. Considering such a pair, let $a_1, a_2, \ldots, a_n \in \{0,1\}$ denote the values of commitments placed on the first row (in the pair) and $b_1, b_2, \ldots, b_n \in \{0,1\}$ denote those of commitments on the second row.

1. V attaches the corresponding number card to each of the $4n$ cards.
2. V applies the "input-preserving" Mizuki–Sone XOR protocol obtained by Sections 3.1.4 and 3.1.6 to the commitments to a_i and b_i to produce a commitment to $a_i \oplus b_i$ for every i, $1 \leq i \leq n$. Note that V will return the $4n$ cards to their original positions after the next step.

3. V uses the "input-preserving" Mizuki–Sone OR protocol obtained by Sections 3.1.3 and 3.1.6[5] exactly $n-1$ times to reveal the value of $\bigvee_{j=1}^{n}(a_j \oplus b_j)$. If it is 0, it means $a_i = b_i$ for every i, and hence, V rejects it.

Equality Verification phase. In this phase, V verifies that the Equality rule is satisfied.

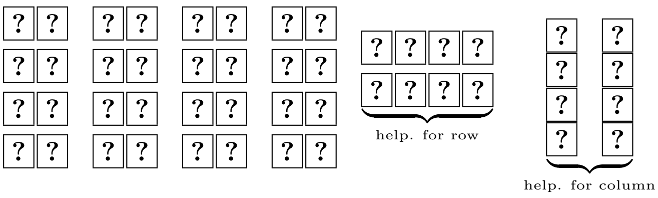
1. For every row, V repeats the following.
   a. V attaches the corresponding number card to each of the $2n$ cards.
   b. V applies a pile scramble shuffle.
   c. V reveals the resulting n commitments. If the number of commitments to 0 is not equal to that of commitments to 1, V rejects it.
   d. Similar to the input-preserving function evaluation technique shown in Section 3.1.6, V returns the n commitments to their original positions.
2. For every column, V follows the same steps except for Steps (a) and (d). Since the n commitments will not be used after this phase, V does not need to return them to their original positions.

This protocol uses n^2 black cards, the same number of red cards, and $4n$ number cards (recall that we have an $n \times n$ Takuzu grid). The numbers of required shuffles are $4n(n-2)$ in the Adjacent Verification phase, $2n^2(n-1)$ in the Uniqueness Verification phase, and $3n$ in the Equality Verification phase.

3.2.2 Protocol 2: Verifying All the Constraints Simultaneously

Protocol 2 verifies that all the constraints are satisfied simultaneously using helping cards that will be placed in the Setup phase. When displaying a figure, we are given a 4×4 Takuzu grid as an example.

Setup phase. The prover P places a commitment to each cell according to the solution. In addition, to show that all the constraints are satisfied, P arranges face-down sequences corresponding to all the sequences in $\mathsf{tkz}(n)$ except for those in the solution (for both row and column):



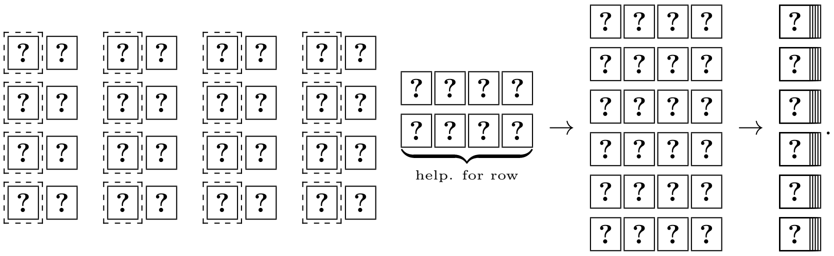
,

where a black card ♣ corresponds to 0 and a red card ♡ corresponds to 1 in any helping sequence for the row, and ♡ corresponds to 0 and ♣ corresponds to 1 in any helping sequence for the column. As shown in Table 1, the number of such helping sequences is two in each direction in this case of 4×4 grid.

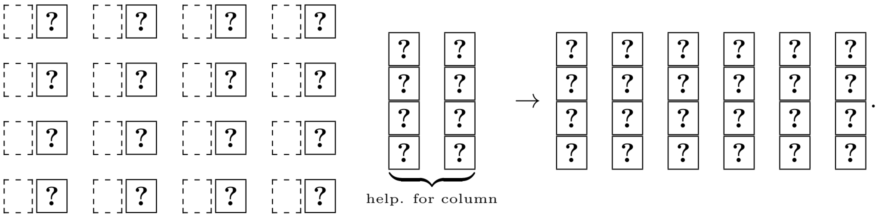
[5] For the two additional cards, we can make use of any two revealed cards appearing in the previous step without opening the number cards.

Verification phase. In this phase, V verifies all the constraints, namely the Equality, Uniqueness, and Adjacent rules by revealing the commitments along with the helping sequences after applying a pile-scramble shuffle. Note that V can also verify that the commitments placed by P in the Setup phase form the valid ones according to the encoding rule (1) (e.g., not $\boxed{\clubsuit}\boxed{\clubsuit}$ or $\boxed{\heartsuit}\boxed{\heartsuit}$).

1. For all the rows, take the left card of each commitment to make n sequences (along with the helping sequences for the rows).



2. Apply a pile-scramble shuffle to the sequence of piles.
3. Reveal the cards of all sequences. If there are either (i) a sequence whose number of black cards is not the same as that of red cards, (ii) two identical sequences, or (iii) a sequence containing more than two consecutive 0s or 1s, then V rejects it.
4. For all the columns, take the right card of each commitment to make n sequences (along with the helping sequences for the columns).



Then, the same is done.

This protocol uses $n \cdot |\mathsf{tkz}(n)|$ black cards and the same number of red cards when we have an $n \times n$ Takuzu grid. See Table 1 again for the value of $|\mathsf{tkz}(n)|$. The number of required shuffles is two.

3.3 Comparison

Let us compare the two protocols for Takuzu presented in the previous subsection. Table 2 summarizes the numbers of required cards and shuffles for the protocols.

Table 2 The numbers of required cards and shuffles for Protocols 1 and 2 when we have an $n \times n$ Takuzu grid such that n is up to eight.

	#Cards			#Shuffles		
	$n = 4$	$n = 6$	$n = 8$	$n = 4$	$n = 6$	$n = 8$
Protocol 1	48	96	160	140	474	1112
Protocol 2	48	168	544	2	2	2

According to this table, there is a trade-off between the numbers of required cards and shuffles, i.e., Protocol 1 presented in Section 3.2.1 needs a less number of cards but needs a more number of shuffles than Protocol 2 presented in Section 3.2.2. Both protocols are reasonable, and hence, P and V may choose their favorite one. Let us stress that pencil puzzles are usually played on a board of small size, say $n = 8$, and also that players enjoying a puzzle normally do not use computers to solve it.

3.4 Security Proofs for Takuzu

We prove the security of our construction. We consider a *shuffle functionality* which is an indistinguishable shuffle of face down cards. The first part is dedicated to give proofs of protocol 1 while the second part is dedicated to prove the security for protocol 2.

3.4.1 Security Proofs of Protocol 1

Takuzu Completeness

We show that if P knows a solution of a given Takuzu grid then he is able to convince V.

Proof. Suppose that P knows a solution S of the initial grid G and runs the input phase described in subsection 3.2.1. Then we show that P is able to perform the proof for the three phases: (AV) adjacent verification phase, (UV) uniqueness and verification phase, and equality verification phase (EV).

Since S is a solution of G, S is a valid grid respecting all the constraints. If S respects the adjacent rule so the six-card trick outputs 0 in all cases. Indeed, if the number are all equals then the rearranging step (step 1 of the six-card trick) has the same output than the input. For example, consider the sequence 101 which is rearrange as:

$$\overset{1}{\boxed{\heartsuit}}\overset{2}{\boxed{\clubsuit}}\overset{3}{\boxed{\clubsuit}}\overset{4}{\boxed{\heartsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}} \rightarrow \overset{1}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}}\overset{3}{\boxed{\clubsuit}}\overset{2}{\boxed{\clubsuit}}\overset{5}{\boxed{\heartsuit}}\overset{4}{\boxed{\heartsuit}}\ .$$

The random cut will keep the pattern, up to a cyclic shift. The same result holds for other possible sequences (there are 6 of them).

We conclude that S succeeds the AV challenge.

We show that S passes the UV challenge. The verification is done toward each possible pair of row (and column) of the grid. Consider two rows where a_i denote the values of commitments on the first row and b_i the values for the second row. Since S is a solution those two rows are different, meaning that there exists at least a value j for which $a_j \neq b_j$. This implies that $a_j = b_j \oplus 1$ (recall that $\forall i = 1 \dots n$ we have $a_i, b_i \in \{0, 1\}$) meaning that $a_j \oplus b_j = 1$. Thus the disjunction of all the possible $a_i \oplus b_i$ will output 1 (since at least on of its term is equal to 1). Repeating this process for each possible pair of rows and columns leads to always output 1 in step 3 of the UV.

Lastly, we show that S succeeds the EV challenge. Since it is a solution there is the same number of 0 and 1 in each row and column. When shuffling the cards, only the their order is modified but not their value thus the equality property still holds.

We conclude that P convinces V for AV, UV and EV phases. ◀

Takuzu Soundness

We show that if P does not provide a solution of a given Takuzu grid then he is not able to convince V with probability 1.

Proof. Suppose that P does not know the solution, we want to show that V will detect it during, at least, one verification phase.

First, notice that if P places a commitment that respects all the Takuzu rules then it is a solution. Thus if at least one rule is not respected then it is not a solution. Hence, we consider three possible cases corresponding to each rule that is not respected:

- If the adjacent rule is not respected, then there exists three consecutive commitments that have the same value (either 0 or 1). Without loss of generality, let consider that those values are all 0's. Thus the the rearrange step is:

  $$\begin{array}{cccccc} 1&2&3&4&5&6\\ \boxed{\clubsuit}&\boxed{\heartsuit}&\boxed{\clubsuit}&\boxed{\heartsuit}&\boxed{\clubsuit}&\boxed{\heartsuit} \end{array} \rightarrow \begin{array}{cccccc} 1&6&3&2&5&4\\ \boxed{\clubsuit}&\boxed{\heartsuit}&\boxed{\clubsuit}&\boxed{\heartsuit}&\boxed{\clubsuit}&\boxed{\heartsuit} \end{array} .$$

  Thus a random cut will keep this alternating pattern. (Note that the same result holds with all 1 but black cards are replaced by red cards and vice-versa.) Hence, the six-card trick outputs 1 so V rejects P's commitments.
- If the uniqueness rule is not respected, then at least two rows or two columns are identical. Thus, for all $i = 1 \dots n$, we have $a_i = b_i \implies a_j \oplus b_j = 0$. This implies that the disjunction of all those terms is equal to 0 so V rejects it.
- If the equality rule is not respected, then there exists a row or column where the number of 0 is not equal to the number of 1. W.l.o.g., consider a row with $\frac{n}{2}+1$ 0-commitment and $\frac{n}{2}-1$ 1-commitment. When applying a pile scramble shuffle the 0-commitment remains 0-commitment, and 1-commitment still remains 1-commitment so V will notice that there is $\frac{n}{2}+1$ 0-commitment and $\frac{n}{2}-1$ 1-commitment. Finally, V won't be convinced. ◀

Zero-knowledge

We show that during the verification process, V learns nothing about P's solution.

Proof. The idea of the proof is described in [13]. Proving zero-knowledge implies to describe an efficient simulator which is an algorithm that simulates any interaction between a cheating verifier and a real prover. The simulator has no access to the correct solution but it has an advantage over the prover: when the cards are shuffled, the simulator can swap the decks with different ones. We thus show how to construct a simulator for each challenge:

Adjacent Verification challenge: The simulator chooses randomly S such that three consecutive cells never contain the same number. Note that the uniqueness and equality rule may not hold. Then it simulates the interaction between the prover and the verifier. For each three vertically (or horizontally) consecutive commitments, the six-card trick outputs 0 (there are exactly two identical number).

Uniqueness Verification challenge: When the verifier checks for pair of rows or columns, the simulator picks cards to form distinct rows or columns (for example, during the Mizuki-Sone XOR shuffle phase).

Equality Verification challenge: During the pile scramble shuffle, the simulator places $\frac{n}{2}$ 0-commitment and $\frac{n}{2}$ 1-commitment in a random order. ◀

We conclude that our protocol for Takuzu is complete, soundness and zero-knowledge.

3.4.2 Security Proofs of Protocol 2

Completeness

We show that if P knows a solution of a given Takuzu grid then he is able to convince V.

Proof. Suppose that P knows a solution S of the initial grid G and runs the input phase described in subsection 3.2.2. Then we show that P is able to perform the proof for the verification phase.

Since S is a solution of G, S is a valid grid respecting all the constraints. Indeed S respects the adjacent rule so each three consecutive commitments cannot be all the same. Thus the left cards of each commitment cannot be the same (recall our encoding 1). The other rules can be verified using the same process since each left card (or right) fully determine the value of a commitment. Indeed, if the left card is $\boxed{\clubsuit}$ the the commitment corresponds to the value 0 and if the left card is $\boxed{\heartsuit}$ then it corresponds to a 1-commitment. We conclude, that if P's commitment corresponds to the solution of G then all the constraints can be verified by V when revealing the commitments. ◀

Soundness

We show that if P does not provide a solution of a given Takuzu grid then he is not able to convince V with probability 1.

Proof. Suppose that P does not know the solution, we want to show that V will detect it during the verification phase.

First, notice that if P places a commitment that respects all the Takuzu rules then it is a solution. Thus if at least one rule is not respected then it is not a solution. Hence, we consider three possible cases corresponding to each rule that is not respected:

- If the adjacent rule is not respected, then there exists three consecutive commitments that have the same value (either 0 or 1). Since the order of the cards is kept (only the pile are shuffled), V can detect when three consecutive cards are identical.
- If the uniqueness rule is not respected, then at least two rows or two columns are identical. Again, V will detect it since all the left (right) cards are revealed and that left (right) cards fully determine a commitment value.
- If the equality rule is not respected, then there exists a row or column where the number of 0 is not equal to the number of 1. As seen in the previous case, V won't be convinced since the number of 0 does not correspond to the number 1. ◀

Zero-knowledge

We show that during the verification process, V learns nothing about P's solution.

Proof. The idea is the same as for protocol 1. We show how to construct a simulator for the challenge. During the pile-scramble phase, the simulator replaces each pile with a sequence of $\mathtt{tkz}(n)$. Thus the set of those sequence verifies the rules. ◀

We conclude that our protocol for Takuzu is complete, soundness and zero-knowledge.

4 Our ZKP Protocol for Juosan

In this section, applying the ideas shown in Section 3, we construct a ZKP protocol for Juosan, which allows the prover P (aka Q) to convince the verifier V (aka James Bond) that he really knows a solution.

4.1 Subprotocol: Five-Card Trick

We introduce the *five-card trick* [8] in this subsection, which is used in our construction to verify Rules 2 and 3.

Given two commitments to $a, b \in \{0,1\}$ (along with a red card $\boxed{\heartsuit}$), the five-card trick [8] outputs $a \wedge b$: $\underbrace{\boxed{?}\boxed{?}}_{a}\ \underbrace{\boxed{?}\boxed{?}}_{b}\ \boxed{\heartsuit} \rightarrow \cdots \rightarrow a \wedge b$. The protocol proceeds as follows.

1. Rearrange the sequence as follows: $\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}} \rightarrow \overset{2}{\boxed{?}}\overset{1}{\boxed{?}}\overset{5}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}$.
2. Apply a random cut to the sequence: $\langle \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \rangle \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$.
3. Reveal the sequence. If the resulting sequence is:
   a. $\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}$ (apart from cyclic shifts), the output is $a \wedge b = 1$.
   b. $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}$ (apart from cyclic shifts), the output is $a \wedge b = 0$.

4.2 Our Construction

We are now ready to present the full description of our ZKP protocol for Juosan. Let us consider that we are given a 5×5 Juosan grid as an example.

Our construction consists of three phases, the Setup phase, Adjacent Verification phase, and Room Verification phase.

Setup phase. Regarding a vertical dash (|) as 0 and a horizontal dash (—) as 1, the prover P places a commitment to each cell according to the solution:

$$\begin{array}{ccccc}
\boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} \\
\boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} \\
\boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} \\
\boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} \\
\boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?} & \boxed{?}\boxed{?}
\end{array}.$$

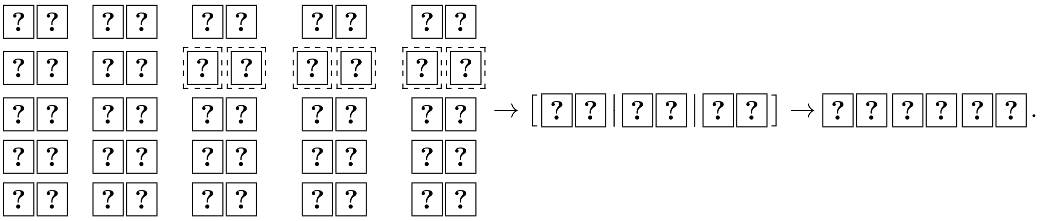
Adjacent Verification phase. In this phase, V repeats applications of the Mizuki–Sone AND protocol [24] and five-card trick [8] enhanced by the input-preserving function evaluation technique to verify that the Adjacent condition is satisfied. Note that V can also verify that the commitments placed by P in the Setup phase form the valid ones according to the encoding rule (1).

1. Let us verify that there are no three consecutive horizontal dashes in any column. The fact that three horizontal dashes are not consecutive to the vertical means that there is at least one vertical dash among them. Therefore, it suffices to confirm the AND value of of the corresponding three commitments is false because a vertical dash is encoded as 0 and a horizontal dash as 1.

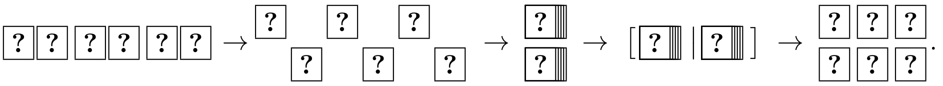
   Let $a, b, c \in \{0,1\}$ be the values of commitments on three consecutive cells in a column. First, for commitments to a and b, perform the Mizuki–Sone AND protocol described in Section 3.1.3. Then, a commitment to $a \wedge b$ is obtained.
2. Perform the five-card trick described in Section 4.1 for the commitments to $a \wedge b$ and c. If the five-card trick outputs 1, V rejects it.
3. Restore commitments to a, b, and c by the input-preserving function evaluation technique described in Section 3.1.6.
4. The same is done for rows. In this case, let the encoding be reversed.

Room Verification phase. In this phase, V verifies the Room rule by revealing the commitments after applying pile-scramble shuffles.

1. Apply a pile-scramble shuffle to all commitments in a territory with a number:



2. Take all the left cards and all the right cards of these commitments to make two piles. Then, apply a pile-scramble shuffle to the two piles:



3. Reveal all the cards of the piles. If the number of black cards or red cards is not the same as the number written on the territory, V rejects it. For example, in the case of a 3-cell territory with a number "3," each of the following two types of card groups should appear with a probability of 1/2: $\begin{array}{c}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}\\ \boxed{\clubsuit}\boxed{\clubsuit}\boxed{\clubsuit}\end{array}$, $\begin{array}{c}\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\clubsuit}\\ \boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}\end{array}$, where the order of cards in the card set does not matter.
4. The same is done for all other numbered territories.

The numbers of required shuffles are $3(m(n-2)+n(m-2))$ in the Adjacent Verification phase and k in the Room Verification phase when we have an $m \times n$ Juosan grid and k territories. This protocol uses $mn+1$ black cards, the same number of red cards, and eight number cards.

4.3 Optimized Adjacent Verification for Juosan

In the original Adjacent Verification phase of our protocol for Juosan presented in Section 4.2, the AND value $a \wedge b \wedge c$ for $a, b, c \in \{0,1\}$ is securely computed to show the validity of three consecutive commitments. We present an optimization technique to show the validity of four consecutive commitments as follows.

1. Let $a, b, c, d \in \{0,1\}$ be commitments of four consecutive cells in a column. First, for commitments to b and c, perform the Mizuki–Sone AND protocol described in Section 3.1.3. Then, a commitment to $b \wedge c$ is obtained.
2. Let $x_1 = b \wedge c$, $x_2 = a$, and $c_3 = d$. By slightly modifying the Mizuki–Sone AND protocol, the following protocol is obtained:

$$\underbrace{\boxed{?}\boxed{?}}_{x_1}\ \underbrace{\boxed{?}\boxed{?}}_{x_2}\ \boxed{\clubsuit}\boxed{\heartsuit}\ \underbrace{\boxed{?}\boxed{?}}_{x_3}\ \boxed{\clubsuit}\boxed{\heartsuit} \rightarrow \cdots \rightarrow \underbrace{\boxed{?}\boxed{?}}_{x_1 \wedge x_2}\ \underbrace{\boxed{?}\boxed{?}}_{x_1 \wedge x_3}.$$

Note that this uses one random bisection cut only. Then, two commitments of $x_1 \wedge x_2 = a \wedge b \wedge c$ and $x_1 \wedge x_3 = b \wedge c \wedge d$ are obtained.
3. Open the commitments of $a \wedge b \wedge c$ and $b \wedge c \wedge d$. If they are not $(0,0)$, V rejects it.
4. Obtain the commitments to a, b, c, and d by the input-preserving function evaluation technique described in Section 3.1.6.

4.4 Security Proofs for Juosan

We prove the security of our construction. We consider a *shuffle functionality* which is an indistinguishable shuffle of face down cards.

Juosan Completeness

We show that if P knows a solution of a given Takuzu grid then it is able to convince V.

Proof. Suppose that P knows a solution S of the initial grid G and runs the setup phase described in Section 4. Then we show that P is able to perform the proof for the two phases: adjacent verification phase (AV) and room verification phase (RV).

Since S is a solution of the grid G, we show that S is a valid grid respecting all the constraints.

We first consider the adjacent verification. Let us take an example, the other cases (here 8 possible cases) are done the same way. We consider the case of horizontal dashes in a column for verifying the adjacent (horizontal) rule. We need to show that the AND value of these commitments is not equal to 1. Note that if we inverse the encoding rule ($\boxed{\heartsuit}\boxed{\clubsuit} = 0$ and $\boxed{\clubsuit}\boxed{\heartsuit} = 1$) we can verify that no three consecutive vertical dashes are placed in a given row.

We consider the 101-commitment: $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\clubsuit}$.

First we take the first four cards and apply the Mizuki-Sone AND protocol:

$$\overset{1}{\boxed{\heartsuit}}\overset{2}{\boxed{\clubsuit}}\overset{3}{\boxed{\clubsuit}}\overset{4}{\boxed{\heartsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}} \rightarrow \overset{1}{\boxed{\heartsuit}}\overset{3}{\boxed{\clubsuit}}\overset{4}{\boxed{\heartsuit}}\overset{2}{\boxed{\clubsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}}.$$

Then the random cut will output two possible combinations:

$$\overset{1}{\boxed{\heartsuit}}\overset{3}{\boxed{\clubsuit}}\overset{4}{\boxed{\heartsuit}}\overset{2}{\boxed{\clubsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}} \text{ or } \overset{2}{\boxed{\clubsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}}\overset{1}{\boxed{\heartsuit}}\overset{3}{\boxed{\clubsuit}}\overset{4}{\boxed{\heartsuit}}.$$

Both cases has output $\boxed{\clubsuit}\boxed{\heartsuit}$ which is simply 0.

Note that if we replace the second commitment by 1 (which is encoded as $\boxed{\heartsuit}\boxed{\clubsuit}$) then after the random cut we have the two possible outputs: $\overset{1}{\boxed{\heartsuit}}\overset{3}{\boxed{\heartsuit}}\overset{4}{\boxed{\clubsuit}}\overset{2}{\boxed{\clubsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}}$ or $\overset{2}{\boxed{\clubsuit}}\overset{5}{\boxed{\heartsuit}}\overset{6}{\boxed{\clubsuit}}\overset{1}{\boxed{\heartsuit}}\overset{3}{\boxed{\heartsuit}}\overset{4}{\boxed{\clubsuit}}$.

The output is $\boxed{\heartsuit}\boxed{\clubsuit}$ which is simply 1 (and this corresponds with the expected value).

Next, we compute the five-card trick for input $\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}$.

The rearrange step outputs $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\clubsuit}$ which is the same pattern of alternating figure meaning that $a \wedge b = 0$. Note that a random cut will not modify the shape of the pattern.

The same process is applied to all other commitments so we can conclude that S respects the adjacent verification for horizontal and vertical dashes. Hence S succeeds the AV challenge.

Note that we can verify the adjacent rule by looking at three consecutives cells and the next three consecutives cells (that is cells a, b, c and then cells b, c, d) or directly apply the optimized adjacent verification in Section 4.3.

S also succeeds the room verification. Indeed, we make two piles corresponding to left cards of each commitment and right cards of each commitment. Thus each vertical dash (encoded as $\boxed{\clubsuit}\boxed{\heartsuit}$) adds a card $\boxed{\clubsuit}$ in a pile and a card $\boxed{\heartsuit}$ in the other pile. Hence, a pile represents the number of vertical dashes while the other represents the number of horizontal dashes (but those two piles are indistinguishable). It remains to count the number of cards that forms the majority to deduce if the room rule is achieved. Finally S is a correct solution for RV challenge.

We conclude that P convinces V for AV phase and for RV phase. ◀

Juosan Soundness

We show that if P does not provide a solution of a given Juosan grid then it is not able to convince V.

Proof. Suppose that P is able to convince V meaning that P can provide S which succeeds AV challenge and RV challenge. We want to show that P knows a solution to Juosan grid G.

During the input phase, P places a commitment.

Since P is able to perform the proof of AV challenge and RV challenge we have: initial cells are the same as in S, horizontal bars are not arranged three times in a column, vertical bars are not arranged three times in a row, and a room has correct numbers of vertical or horizontal bars corresponding to its number.

We deduce that S is a solution of G (since each rule is respected). Hence if P does not provide a solution of G then it fails the proof for at least one challenge. Since those two phases are perform during the proof, P receives two challenges (AV and RV) out of two possibilities.

Hence, if P gives a wrong grid then at least one of those two challenges will fail.

Thus P cannot convince V with a wrong proposition. ◀

Juosan Zero-knowledge

We show that during the verification process, V learns nothing about P's solution.

Proof. We follow the same process as for the zero-knowledge of Takuzu protocol. We thus show how to construct a simulator for each challenge:

Adjacent Verification challenge: The simulator chooses randomly S. Before the final output of the five-card trick, the simulator always chooses a deck for which red and black cards are alternated. Thus the output is always 0 meaning that the Adjacent Verification challenge succeed. Since S was chosen randomly then simulated proofs and real proofs are indistinguishable.

Room Verification challenge: When the verifier checks for vertical direction, the simulator looks at the room number to form the corresponding number with red cards (or black ones) for each piles. This step is done the same way for all rooms. Since each row (or column) are different from one to another, the simulated proofs and real proofs are indistinguishable. ◀

We conclude that our protocol for Juosan is complete, soundness and zero-knowledge.

5 Conclusion

In this paper we improved the existing interactive zero-knowledge proof for Takuzu. Our protocols use a reasonable number of cards and shuffles, implying that they are easy to implement by humans. Our protocols are designed in such a way that the proof is completely sound meaning that a prover P convinces the verifier V with probability 1 if P has a solution. We also proposed an adapted version of this protocol for the Juosan puzzle which had never been proposed before. An interesting puzzle, called *Suguru*, can also be studied with this technique.

References

1 József Balogh, János A. Csirik, Yuval Ishai, and Eyal Kushilevitz. Private computation using a PEZ dispenser. *Theor. Comput. Sci.*, 306(1-3):69–84, 2003.

2 Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 1988. doi:10.1007/0-387-34799-2_4.

3 Marzio De Biasi. Binary puzzle is NP-complete, July 2012. URL: http://www.nearly42.org/vdisk/cstheory/binaryp.pdf.

4 Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 103–112, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/62212.62222.

5 Xavier Bultel, Jannik Dreier, Jean-Guillaume Dumas, and Pascal Lafourcade. Physical zero-knowledge proofs for Akari, Takuzu, Kakuro and KenKen. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, volume 49 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FUN.2016.8.

6 Xavier Bultel, Jannik Dreier, Jean-Guillaume Dumas, Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Atsuki Nagao, Tatsuya Sasaki, Kazumasa Shinagawa, and Hideaki Sone. Physical zero-knowledge proof for Makaro. In *SSS 2018 - 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume 11201 of *Lecture Notes in Computer Science*, pages 111–125, Tokyo, Japan, November 2018. Springer. doi:10.1007/978-3-030-03232-6_8.

7 Yu-Feng Chien and Wing-Kai Hon. Cryptographic and physical zero-knowledge proof: From Sudoku to Nonogram. In Paolo Boldi and Luisa Gargano, editors, *Fun with Algorithms 2010*, volume 6099 of *LNCS*, pages 102–112. Springer, 2010.

8 Bert den Boer. More efficient match-making and satisfiability the five card trick. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 208–217, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

9 Jannik Dreier, Hugo Jonker, and Pascal Lafourcade. Secure auctions without cryptography. In *Fun with Algorithms, 7th International Conference, FUN'14*, pages 158–170, 2014. doi:10.1007/978-3-319-07890-8_14.

10 Jean Guillaume Dumas, Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Tatsuya Sasaki, and Hideaki Sone. Interactive physical zero-knowledge proof for Norinori. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11653 LNCS:166–177, 2019. doi:10.1007/978-3-030-26176-4_14.

11 Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–189, 1996. doi:10.1007/s001459900010.

12 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. Knowledge complexity of interactive proof-systems. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 291–304, 1985. doi:10.1145/3335741.3335750.

13 Ronen Gradwohl, Moni Naor, Benny Pinkas, and Guy N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of Sudoku puzzles. In *Proceedings of the 4th International Conference on Fun with Algorithms*, FUN'07, pages 166–182, Berlin, Heidelberg, 2007. Springer-Verlag.

14 James Heather, Steve A. Schneider, and Vanessa Teague. Cryptographic protocols with everyday objects. *Formal Aspects of Computing*, 26:37–62, 2013.

15 Rie Ishikawa, Eikoh Chida, and Takaaki Mizuki. Efficient card-based protocols for generating a hidden random permutation without fixed points. In Cristian S. Calude and Michael J. Dinneen, editors, *UCNC 2015*, volume 9252 of *LNCS*, pages 215–226. Springer, 2015.

16 Chuzo Iwamoto and Tatsuaki Ibusuki. Kurotto and Juosan are NP-complete. In *The 21st Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCG3 2018)*, pages 46–48, Ateneo de Manila University, Philippines, September 2018.

17 Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Tatsuya Sasaki, and Hideaki Sone. A physical ZKP for Slitherlink: How to perform physical topology-preserving computation. In Swee-Huay Heng and Javier Lopez, editors, *Information Security Practice and Experience*, pages 135–151, Cham, 2019. Springer International Publishing.

18 Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

19 Daiki Miyahara, Tatsuya Sasaki, Takaaki Mizuki, and Hideaki Sone. Card-based physical zero-knowledge proof for Kakuro. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E102.A(9):1072–1078, 2019. doi:10.1587/transfun.E102.A.1072.

20 Takaaki Mizuki. Efficient and secure multiparty computations using a standard deck of playing cards. In *Cryptology and Network Security*, pages 484–499, November 2016. doi:10.1007/978-3-319-48965-0_29.

21 Takaaki Mizuki, Yoshinori Kugimoto, and Hideaki Sone. Secure multiparty computations using a dial lock. In Jin-yi Cai, S. Barry Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China*, volume 4484 of *LNCS*, pages 499–510. Springer, May 2007. doi:10.1007/978-3-540-72504-6_45.

22 Takaaki Mizuki, Yoshinori Kugimoto, and Hideaki Sone. Secure multiparty computations using the 15 puzzle. In Andreas W. M. Dress, Yinfeng Xu, and Binhai Zhu, editors, *Combinatorial Optimization and Applications, First International Conference, COCOA 2007, Xi'an, China*, volume 4616 of *LNCS*, pages 255–266. Springer, August 2007. doi:10.1007/978-3-540-73556-4_28.

23 Takaaki Mizuki and Hiroki Shizuya. Practical card-based cryptography. In *Fun with Algorithms, 7th International Conference, FUN'14*, pages 313–324, 2014. doi:10.1007/978-3-319-07890-8_27.

24 Takaaki Mizuki and Hideaki Sone. Six-card secure AND and four-card secure XOR. In Xiaotie Deng, John E. Hopcroft, and Jinyun Xue, editors, *Frontiers in Algorithmics, Third International Workshop, FAW 2009, Hefei, China, June 20-23, 2009. Proceedings*, volume 5598 of *LNCS*, pages 358–369. Springer, 2009.

25 Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 285–297. Springer, 2005.

26 Tal Moran and Moni Naor. Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006*, volume 4004 of *LNCS*, pages 88–108. Springer, 2006. doi:10.1007/11761679_7.

27 Tal Moran and Moni Naor. Split-ballot voting: everlasting privacy with distributed trust. *ACM Trans. Inf. Syst. Secur.*, 13:246–255, 2010. doi:10.1145/1315245.1315277.

28 Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Pile-shifting scramble for card-based protocols. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 101(9):1494–1502, 2018.

29 Tatsuya Sasaki, Takaaki Mizuki, and Hideaki Sone. Card-based zero-knowledge proof for Sudoku. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy*, volume 100 of *LIPIcs*, pages 29:1–29:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FUN.2018.29.

30 Kazumasa Shinagawa and Takaaki Mizuki. The six-card trick: Secure computation of three-input equality. In Kwangsu Lee, editor, *Information Security and Cryptology – ICISC 2018*, volume 11396 of *LNCS*, pages 123–131, Cham, 2019. Springer.

31 Kazumasa Shinagawa, Takaaki Mizuki, Jacob C. N. Schuldt, Koji Nuida, Naoki Kanayama, Takashi Nishide, Goichiro Hanaoka, and Eiji Okamoto. Multi-party computation with small shuffle complexity using regular polygon cards. In Man Ho Au and Atsuko Miyaji, editors, *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, volume 9451 of *LNCS*, pages 127–146. Springer, 2015. doi:10.1007/978-3-319-26059-4_7.

32 Kazumasa Shinagawa and Koji Nuida. A single shuffle is enough for secure card-based computation of any circuit. *IACR Cryptology ePrint Archive*, pages 1–19, 2019.

33 Itaru Ueda, Daiki Miyahara, Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Secure implementations of a random bisection cut. *International Journal of Information Security*, 19:445–452, August 2019. doi:10.1007/s10207-019-00463-w.

34 Putranto Hadi Utomo and Ruud Pellikaan. Binary puzzles as an erasure decoding problem. In *Proceedings of the 36th WIC Symposium on Information Theory in the Benelux*, pages 129–134, 2015. URL: www.win.tue.nl/~ruudp/paper/72.pdf.

Speeding up Networks Mining via Neighborhood Diversity

Gennaro Cordasco
Dipartimento di Psicologia, Università della Campania “Luigi Vanvitelli”, Caserta, Italy
gennaro.cordasco@unicampania.it

Luisa Gargano
Dipartimento di Informatica, Università di Salerno, Fisciano, Italy
lgargano@unisa.it

Adele A. Rescigno
Dipartimento di Informatica, Università di Salerno, Fisciano, Italy
arescigno@unisa.it

Abstract

Parameterized complexity was classically used to efficiently solve NP-hard problems for small values of a fixed parameter. Then it has also been used as a tool to speed up algorithms for tractable problems. Following this line of research, we design algorithms parameterized by neighborhood diversity (nd) for several graph theoretic problems in P (e.g., Maximum Matching, Triangle counting and listing, Girth and Global minimum vertex cut). Such problems are known to admit algorithms parameterized by modular-width (mw) and consequently – being the nd a “special case” of mw – by nd. However, the proposed novel algorithms allow to improve the computational complexity from a time $O(f(\mathsf{mw}) \cdot n + m)$ – where n and m denote, respectively, the number of vertices and edges in the input graph – which is multiplicative in n to a time $O(g(\mathsf{nd}) + n + m)$ which is additive only in the size of the input.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Neighborhood Diversity, Maximum Matching, Triangle Counting, Girth, Global minimum vertex cut

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.21

1 Introduction

A large online marketplace *Grove* is about to design a novel marketing strategy exploiting the data available thanks to their largely adopted premium program *Grove-FUN*. The CEO of Grove realized that customers' activities, reactions, and interactions on *Grove-FUN* can be used to perform predictive analysis on marketing, which increases both customer satisfaction and company returns. Indeed, the predictive analysis can be used to devise:

- Personalized recommendation system. For example, when a user adds a comic to his/her online shopping cart, similar comics purchased by other customers or other products purchased by customers having some similarity with the user can be recommended.
- Anticipatory Shipping Model for predicting the products customers are going to purchase, when and where they might need the products. According to the analysis, the items are pushed to a local distribution center or warehouse so they will be ready for shipping once a customer orders them. This approach increases product sales and profit margins because it reduces delivery time and overall expenses.
- Price optimization. Prices are set according to customer activities, item preferences, order history, and other factors.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 21; pp. 21:1–21:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Viral marketing strategies. Sales of a new product can be improved by taking advantage of the human tendency to conform [5] by means of a viral marketing campaign based on targeted discounts [10].

The Grove Marketing Analysts realized that the data available on the premium program Grove-FUN can be easily modeled as a networked structure in terms of nodes (customers) and edges, or links (friendship or interactions) that connect them. This approach enables them to study the data through the use of networks and graph theory [16]. Several classical graph theory algorithms gained popularity in social network analysis. For instance, Triangle counting is used to detect communities and measure the cohesiveness of those communities, Maximum matching can be used to devise matching market strategies, the betweenness centrality is used to individuate influential vertices and their importance. Unfortunately, the Grove-FUN network is very large, with millions of customers and billions of edges. It is often prohibitively expensive to perform network analysis using standard algorithms on very large networks.

Fortunately, the clever Grove analysts have observed that the structure of Grove-FUN relationships follows the rules of social relationships. For instance, it is possible to observe how the connections are balanced between staying within a well identified community (where everybody knows each other) and cutting across communities. Such a specific structure of the network suggests a different way of thinking about standard algorithms in terms of their dense communities, and the ways in which they interact with each other.

On the basis of this observation, the following mathematical model was put forward. The network of customers is represented by a graph $G = (V, E)$, where V is the set of customers, and there is an edge between two customers if they reviewed the same product. Then the analysts were able to compress the network (keeping most of the information described by the original network) – exploiting the neighborhood diversity approach, introduced by Lampis in [32] – grouping similar nodes. A new graph H, called the type graph of G, characterized by a smaller number of nodes (each representing a group of customers) is generated in linear time and, starting from it, analysts have started having FUN redesigning and speeding up some classical algorithms such as maximum matching, triangle counting, girth, and global minimum vertex cut to use the graph H solving problems defined on G.

2 The hardness in P context

Algorithmic research aims to determine the best possible running time algorithms for computational problems. A first goal toward the classification of the problems according to their complexity was achieved by the NP-completeness theory, whose goal is to identify problems that are unlikely to be solved in polynomial time. However, there are still many intensively studied problems in P for which the worst-case running time of the best current algorithm is not known to be optimal. Namely, many important problems have classical algorithms running in $\tilde{O}(n^k)$ time[1] for some constant k, and this running time has not been significantly improved upon, in spite on many years of intensive research.

Recently, the *Hardness in P* tool for determine a hierarchy of the complexity of polynomial-time solvable problems has been introduced [37]. The key starting point here is the conjecture that there are problems that do not admit algorithms that perform significantly better than the already known ones. In particular, it has been conjectured that there is no $O(n^{2-\epsilon})$ time algorithm for 3-SUM and no $O(n^{3-\epsilon})$ time algorithm for All-Pairs Shortest Paths. Moreover,

[1] The $\tilde{O}(f)$ notation ignores factors of $\log(f)$

the *Hardness in P* theory is based on the Strong Exponential Time Hypothesis (SETH): There is no $c < 2$ such that k-SAT can be solved in time $O(c^n)$ for each fixed k. This conjecture is used to obtain lower bounds to the complexity of some problems in P, in the sense that the existence of a faster algorithm for one of these problems implies the existence of a faster algorithm for one of the fundamental problems mentioned above. Recent work in the area can be found in [1, 2, 9, 22, 38].

On the positive side, efforts have been made to improve algorithms for problems in P on some restricted classes of graphs. In particular, parameterized algorithms have been recently proposed as a tool to speed up algorithms for problems in P [12]. Parameterized complexity was classically used to efficiently solve NP-hard problems for small values of a fixed parameter [14, 36]. Formally, a parameterized problem with input size n and parameter t is called *fixed parameter tractable (FPT)* if it can be solved in time $f(t) \cdot n^c$, where f is a function only depending on t and c is a constant.

Unfortunately there are several parameters whose computation is an NP-hard problem itself. As an example computing treewidth, rankwidth, and vertex cover are all NP-hard problems– even though they are computable in FPT time when their respective parameters are bounded. Moreover, the parameterized complexity of computing the clique-width of a graph is still an open problem [13]. On the contrary, modular-width [21] and neighborhood diversity [32] are two recently introduced parameters that are computable in linear time $O(m)$ on a graph with m edges [3, 7, 11, 12, 15, 18, 19, 21, 24, 25, 32].

2.1 Our results

In this paper we design algorithms parameterized by neighborhood diversity for well studied tractable problems. Namely, we consider the maximum matching, triangle counting, girth, and global minimum vertex cut problems. Such problems are known to admit algorithms parameterized, among other parameters, by modular-width (mw) of the input graph. This implies, being the neighborhood diversity (nd) a "special case" of modular width (i.e., $\mathsf{mw} \leq \mathsf{nd}$)[21], that the same algorithm can be used with respect to nd. However, for a graph with n nodes and m edges the proposed novel algorithms allow to improve the computational complexity from a time $O(f(\mathsf{mw}) \cdot n + m)$, which is multiplicative in n [12] to a time $O(g(\mathsf{nd}) + n + m)$ which is only additive in the size of the input.

3 Neighborhood diversity

Given a graph $G = (V, E)$, two nodes $u, v \in V$ have the same *type* iff $N(v) \setminus \{u\} = N(u) \setminus \{v\}$. The *neighborhood diversity* of a graph G, introduced by Lampis in [32] and denoted by $\mathsf{nd}(G)$, is the minimum number t of sets in a partition $V_1, V_2, \ldots, V_t$, of the node set V, such that all the nodes in V_i have the same type, for $i = 1, \ldots, t$. In the following we use nd, instead of $\mathsf{nd}(G)$, when the graph G is clear from the context. The family $\mathcal{V} = \{V_1, V_2, \ldots, V_t\}$ is called the *type partition* of G.

Let $G = (V, E)$ be a graph with type partition $\mathcal{V} = \{V_1, V_2, \ldots, V_t\}$. By definition, each V_i induces either a *clique* or an *independent set* in G. Whenever $V_i \in \mathcal{V}$ is a singleton, we consider it as inducing an independent set, so any V_i inducing a clique has $|V_i| \geq 2$. For each $V_i, V_j \in \mathcal{V}$, we get that either each node in V_i is a neighbor of each node in V_j or no node in V_i has a neighbor in V_j.

Starting from a graph G and its type partition $\mathcal{V} = \{V_1, \ldots, V_t\}$, we can see each element of $\mathcal{V}$ as a vertex of a new graph H, called the *type graph* of G, with $V(H) = \{1, 2, \ldots, t\}$ and

$$E(H) = \{(x, y) \mid x \neq y, \text{ there is a complete bipartite graph between } V_x \text{ and } V_y \text{ in } G, \}$$
$$\cup \{(x, x) \mid \ |V_x| \geq 2 \text{ and } V_x \text{ and induces a clique in } G\}.$$

Determining $\mathsf{nd}(G)$ and the corresponding type partition, can be done in time $O(n + m)$ [32].

4 Maximum matching

A matching in a graph is a set of edges with pairwise disjoint end vertices. The MAXIMUM MATCHING (MM) problem consists in computing a matching of maximum size.

MAXIMUM MATCHING can be solved in polynomial time by Edmond's algorithm [17]. A simple implementation of such an algorithm requires time $O(n^4)$, however Micali and Vazirani showed how to implement Edmond's algorithm in time $O(m\sqrt{n})$ [34].

MAXIMUM MATCHING is considered fundamental to the study of fixed-parameter algorithm for problems in P [26, 33]. In particular Mertzios et al. designed new algorithms to solve maximum matching in $O(\rho^{O(1)}(n+m))$ time for various graph parameters ρ [33]. Coudert et al. gave a $O(\rho^4 n + m)$ time algorithms for solving MM, when parameterized by either the modular-width or the P_4-sparseness of the graph [12]. Recently, Kratsch et al. improved this last result to $O(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$[31].

In the following we present an algorithm parameterized by neighborhood diversity for solving the MAXIMUM MATCHING of a graph $G = (V, E)$. Our algorithm will use the solution of a generalization of MM, namely the MAXIMUM b-MATCHING problem [20].

Let $b : V \to \mathbb{Z}^+$ be a function on the vertices of G (where G may have also self-loops), a *b-matching* for G is a function $x : E \to \mathbb{Z}^+$ such that for each $u \in V$

$$\sum_{v:\ u\neq v \wedge (u,v)\in E} x(u,v) + 2x(u,u) \leq b(u). \qquad (1)$$

The value $x(u,v)$ represents the multiplicity of the edge (u,v), that is how many times the edge (u,v) is used by the matching. Note that if $b(u) = 1$, for each $u \in V$, then the function x becomes a classical matching for G.

A maximum b-matching for G is a function x for which $\sum_{(u,v)\in E} x(u,v)$ is maximum. In the following we will call by $|x| = \sum_{(u,v)\in E} x(u,v)$ the *size* of the b-matching x.

Gabow showed how to find a maximum b-matching in time $O(\min\{b(V), n\log n\}\ (m+n\log n))$, where $b(V) = \sum_{v\in V} b(v)$ [20].

4.1 The algorithm

We describe now the proposed algorithm.

1. Let $\mathcal{V} = \{V_1, \ldots, V_t\}$ and $H = (V(H), E(H))$ be the type partition and the type graph of G, respectively. Define the function $b_H : V(H) \to \mathbb{Z}^+$ such that $b_H(i) = |V_i|$.
2. Use Gabow's algorithm to find a maximum b_H-matching for H, let it be $x_H : E \to \mathbb{Z}^+$.
3. Construct the desired maximum matching M for G using x_H.

We show now how to implement the above step 3. To this aim, we first notice that (1) implies

$$\sum_{\substack{j\\ i\neq j\wedge(i,j)\in E}} x_H(i,j) + 2x_H(i,i) \leq b_H(i) \qquad \text{for each } i \in V(H)$$

$$x_H(i,j) \leq \min\{b_H(i), b_H(j)\} \qquad \text{for each } (i,j) \in E(H)$$

Fix any $i \in V(H)$ and let $j_1, j_2, \ldots, j_{a(i)}$ be the neighbors of i in H such that $x_H(i, j_\ell) > 0$, for $\ell = 1, \ldots, a(i)$. Recalling that $x_H(i,j)$ represents the multiplicity of the edge (i,j) in the matching and $b_H(i) = |V_i|$, we select

$$s(i) = \sum_{\substack{j\\ i\neq j\wedge(i,j)\in E(H)}} x_H(i,j) + 2x_H(i,i) = \sum_{\ell=1}^{a(i)} x_H(i,j_\ell) + 2x_H(i,i)$$

vertices in V_i and partition them into $a(i)+1$ sets, $S_{i,j_1}, S_{i,j_2}, \dots, S_{i,j_{a(i)}}, C_i$ such that

$$|C_i| = \begin{cases} 2\ x_H(i,i) & \text{if } V_i \text{ induces a clique,} \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad |S_{i,j_\ell}| = x_H(i,j_\ell) \quad \text{for } \ell = 1,\dots,a(i).$$

A matching M for G can be now obtained as

$$M = \left(\bigcup_{\substack{(i,j)\in E(H) \\ i\neq j}} M_{i,j} \right) \bigcup \left(\bigcup_{\substack{i\in V(H) \\ V_i \textit{ is a clique}}} M_i \right), \tag{2}$$

where

- $M_{i,j}$ is a perfect matchings connecting the vertices in $S_{i,j}$ with the vertices in $S_{j,i}$ (recall that $|S_{i,j}| = |S_{j,i}| = x_H(i,j)$);
- M_i is a perfect matchings connecting $x_H(i,i)$ pairs of vertices in C_i, if V_i induces a clique, and $M_i = \emptyset$ otherwise.

▶ **Fact 1.** *The size of the matching M equals that of the b_H-matching in H, that is, $|M| = |x_H|$.*

Proof. By definition $|x_H| = \sum_{(i,j)\in E(H)} x_H(i,j) = \sum_{\substack{(i,j)\in E(H) \\ i\neq j}} |M_{i,j}| + \sum_{i\in V(H)} |M_i| = |M|$. ◀

The following result implies the desired optimality of the matching in (2).

▶ **Lemma 1.** *If x_H is a maximum b_H-matching for H then M is a maximum matching for G.*

Proof. Assume by contradiction that there exists a matching M' for G such that $|M'| > |M|$. Let $y(i,j)$ be the number of edges in M' connecting one vertex in V_i to a vertex in V_j, for $i,j \in V(H)$ (clearly, $y(i,j) = 0$ if $(i,j) \notin E(H)$). Recalling that M' is a matching and, therefore, each vertex in V_i can be the end vertex of only one edge in M', we have that

$$\sum_{j:\ i\neq j \wedge (i,j)\in E(H)} y(i,j) + 2y(i,i) \le |V_i| = b_H(i).$$

By (1), y is a b_H-matching for H. Moreover, its existence contradicts optimality of x_H, since

$$|y| = \sum_{(i,j)\in E(H)} y(i,j) = |M'| > |M| = |x_H|. \qquad ◀$$

Running time. Considering that $b_H(V(H)) = \sum_{i\in V(H)} b_H(i) = \sum_{i\in V(H)} |V_i| = n$ and that $|E(H)| \le \mathsf{nd}^2$, we have that the algorithm due to Gabow [20] for computing a maximum b_H-matching for H requires time $O(\min\{b_H(V(H)), \mathsf{nd}\log\mathsf{nd}\} \cdot (\mathsf{nd}\log\mathsf{nd} + |E(H)|)) = O(\min\{n, \mathsf{nd}\log\mathsf{nd}\} \cdot \mathsf{nd}^2)$.

Summarizing, we have shown the following result.

▶ **Theorem 2.** *For any graph $G = (V,E)$, the maximum matching problem can be solved in time $O(\mathsf{nd}^3 \log \mathsf{nd} + n + m)$.*

We stress that the above algorithm can be generalized to get a maximum b-matching of G. Hence, we can obtain a linear-time kernelization for b-matching (a linear-time algorithm to compress the input into a small input of size $f(\mathsf{nd})$).

5 Cycles

Finding and counting simple cycles in graphs is a classical well studied problem [4]. In particular, triangle detection, counting and/or enumeration problems have applications in many areas, such as spam detection over complex network analysis [6, 35] and bioinformatics [39]. An extensive annotated list of applications can be found in Kolountzakis et al. [29].

Establishing whether there exists a triangle in a general graph is conjectured not to be solvable in time $O(n^{3-\epsilon})$, for $\epsilon > 0$, with a combinatorial algorithm [38]. Furthermore, in [2] it is also conjectured that TRIANGLE COUNTING is not solvable in time $O(n^{\omega-\epsilon})$, for $\epsilon > 0$, with ω being the exponent for fast matrix multiplication[2]. The fastest known algorithm for TRIANGLE COUNTING in general graphs relies on fast matrix multiplication and runs in time $O(n^{\omega})$ [4]. Coudert et al. [12] presented a fast algorithm parameterized by the clique-width cw of the graph running in time $O(\mathsf{cw}^2(n+m))$. Bentert et al. [8] have studied the problem under various parameters including feedback edge number, distance to d-degenerate graphs, and clique-width. They also presented an algorithm for TRIANGLE LISTING in a graph parameterized by the clique-width, running in time $O(\mathsf{cw}^2 \cdot n + n^2 + \#T)$ where $\#T$ denotes the number of triangles in G.

With respect to general cycles in a graph, the GIRTH problem asks to determine the size of the smallest cycle in a given graph. By an old result of Itai and Rodeh [28], if the girth is even, it is possible to determine it in time $O(n^2)$. If, otherwise, the graph has odd girth then any algorithm would have to be able to detect if the graph has a triangle, requiring time $O(n^{\omega})$. Itai and Rodeh also showed that any algorithm that can find a triangle in dense graphs can also compute the girth, so obtaining an $O(n^{\omega})$ time girth algorithm. However, in case of sparse graphs the best running time for the girth is in general $O(nm)$. In [12] was presented a parameterized algorithms that solve the GIRTH problem in time $O(\rho^2(n+m))$ where ρ is either the modular-width or the split-width.

5.1 Triangle counting and listing

Given a graph $G = (V, E)$, TRIANGLE COUNTING problem asks to determine the number of triangles in G. We present an algorithm that solves the TRIANGLE COUNTING problem, then we extend it to solve the TRIANGLE LISTING problem.

Algorithm. Let $\mathcal{V} = \{V_1, V_2, \dots, V_t\}$ be the type partition of G and let $H = (V(H), E(H))$ be the type graph. We count the triangles in G by computing three values

$-a_i$: The number of triangles with all the three vertices in V_i; for $i \in V(H)$.
$-b_i$: The number of triangles with exactly two vertices in V_i; for $i \in V(H)$.
$-c$: The number of triangles with each vertex in a different set of the type partition.

It is immediate to see that $a_i = b_i = 0$ whenever V_i induces an independent set in G. If, otherwise, V_i induces a clique in G then each subset of V_i containing three vertices is a triangle. Furthermore, for each neighbor j of i in H, each pair of vertices in V_i forms a triangle with any vertex in V_j. Hence, for $i \in V(H)$

$$a_i = \begin{cases} \binom{|V_i|}{3} & \text{if } V_i \text{ induces a clique and } |V_i| \geq 3, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

[2] It is known that $2 \leq \omega < 2.3728639$ due to Le Gall [23].

$$b_i = \begin{cases} \binom{|V_i|}{2} \sum_{j:(i,j)\in E(H)} |V_j| & \text{if } V_i \text{ induces a clique and } |V_i| \geq 2 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

To compute c we have to count the number of triangles with each vertex in a distinct set of the type partition. Hence, for each triangle in H involving for instance the vertices $i, j, k \in V(H)$ we have $|V_i||V_j||V_k|$ triangles in G

$$c = \sum |V_i||V_j||V_k|, \tag{5}$$

where the sum is over all $i, j, h \in V(H)$ forming a triangle in H.

We use a result due to Kratsch et al.[31] to compute c in (5). We present it in terms of the type partition of G.

▶ **Lemma 3.** *[31] Let $G = (V, E)$ be a graph with type partition $V_1, V_2, \ldots, V_t$ and type graph H. Consider the weight function $w : E(H) \to R^+$ with $w(i, j) = \sqrt{|V_i||V_j|}$. Let A be the weighted adjacency matrix of H with respect to w. Then, the number of triangles in G with each vertex in a different set of the type partition is*

$$c = \sum_{i,j\in V(H)} \frac{1}{3}(A^2 \circ A)_{i,j} \tag{6}$$

where $A \circ B$ denotes the Hadamard product of the matrices A and B, i.e., $(A \circ B)_{i,j} = A_{i,j} B_{i,j}$.

By (3), (4) and (6), we have that the number of triangles in G is

$$\sum_{\substack{i\in V(H):\\ V_i \text{ induces a clique}\\ |V_i|\geq 3}} \binom{|V_i|}{3} + \sum_{\substack{i\in V(H):\\ V_i \text{ induces a clique}\\ |V_i|\geq 2}} \left(\binom{|V_i|}{2} \sum_{j:(i,j)\in E(H)} |V_j| \right) + \sum_{i,j\in V(H)} \frac{1}{3}(A^2 \circ A)_{i,j} \tag{7}$$

Running time. The first two terms of (7) can be computed in time $O(|E(H)|) \leq O(\mathsf{nd}^2)$. The time to evaluate the last term depends on the time to compute the matrix $A^2 \circ A$. The best algorithm to compute the matrix multiplication A^2 requires time $O(\mathsf{nd}^\omega)$. Finally, multiplying each element of A^2 with the correspondent element in A and summing up all the obtained values takes time $O(\mathsf{nd}^2)$.

▶ **Theorem 4.** *The* TRIANGLE COUNTING *problem can be solved in time $O(\mathsf{nd}^\omega + n + m)$.*

By exploiting the above algorithm and (7) we can easily obtain an algorithm that lists all the triangles of G and so solving the TRIANGLE LISTING problem.

▶ **Theorem 5.** *For any graph $G = (V, E)$, the* TRIANGLE LISTING *problem can be solved in time $O(\mathsf{nd}^\omega + n + m + \#T)$, where $\#T$ denotes the number of triangles in G.*

5.2 Girth

We present an algorithm that finds the girth $\mu(G)$ of any connected graph G.

Algorithm. Let $\mathcal{V} = \{V_1, V_2, \dots, V_t\}$ be the type partition of G and let $H = (V(H), E(H))$ be the type graph.

One can obtain the girth of G by distinguishing the following cases:

1. If there exists $i \in V(H)$ such that $|V_i| \geq 2$ and V_i induces a clique then $\mu(G) = 3$.
2. If Cases 1. does not hold then compute the girth of H $(h = \mu(H))$:
   - If either there exist two neighbors $i, j \in V(H)$, with $|V_i| \geq 2$ and $|V_j| \geq 2$ or there exists a vertex $i \in V(H)$ with $|V_i| \geq 2$ having at least two neighbors then $\mu(G) = \min\{4, h\}$.
   - Otherwise, $\mu(G) = h$.

The algorithm first checks if there is a triangle involving an edge connecting two nodes in the same type set. Indeed, if there exists at least a type set V_i inducing a clique (recall that $|V_i| \geq 2$ in this case) then any two vertices in V_i with any neighbor inside or outside V_i form a triangle (recall that G is connected). If this is the case the algorithm returns $\mu(G) = 3$.

Otherwise, we know that *all the type sets* $V_1, V_2, \dots, V_t$ *induce independent sets.* In this case, any cycle in G must involve only edges between pairs of type sets, and so edges of H. The algorithm computes the girth h of H. Then if there is a cycle of length 4 involving two nodes in the same type set, the algorithm returns the value $\min\{4, h\}$. Cycles of length 4 are identified by the following conditions. If there exist $(i, j) \in E(H)$, with $|V_i| \geq 2$ and $|V_j| \geq 2$, then any two vertices in V_i and any two vertices in V_j induces a cycle of length 4 in G. Analogously, if there exists a vertex $i \in V(H)$ with $|V_i| \geq 2$ having at least two neighbors, say $j, h \in V(H)$, then a vertex in V_j together with a vertex in V_h and any two vertices in V_i induce a cycle of length 4 in G.

If none of the above cases holds, then we know that for each $i \in V(H)$, such that $|V_i| \geq 2$, i has only one neighbor j such that $|V_j| = 1$. Hence, no vertex in V_i may be a vertex of a cycle. Therefore, there is a cycle of length ℓ in G iff there is a cycle of length ℓ in H. Hence, $\mu(G) = \mu(H)$ and the girth of G is obtained by computing the girth of H.

Running time. The worst case in the algorithm is the calculus of the girth of H. Hence, the running time of the algorithm is $O(\mathsf{nd}^{\omega} + n + m)$.

▶ **Theorem 6.** *For any graph $G = (V, E)$, the* GIRTH *problem can be solved in time $O(\mathsf{nd}^{\omega} + n + m)$.*

6 Global minimum vertex cut

In this section we consider the GLOBAL MINIMUM VERTEX CUT problem, a generalization to vertex capacities of the vertex connectivity of a graph.

Given a graph $G = (V, E)$, a set $X \subseteq V$ is a *vertex cut* of G if $G - X$ is disconnected. It is then possible to partition $V(G) - X$ into two non empty sets A_X and B_X where each vertex in A_X has only neighbors in $A_X \cup X$ and, each vertex in B_X has only neighbors in $B_X \cup X$. We call the pair (A_X, B_X) the *disconnected partition* of $G - X$. Given a capacity function $c : V \to R^+$ on the vertices of G, a vertex cut X of minimum capacity $c(X) = \sum_{u \in X} c(u)$ is said the *global minimum vertex cut* of G. The GLOBAL MINIMUM VERTEX CUT problem asks to find the global minimum vertex cut of G given a capacity function c.

As highlighted in [31], the global minimum vertex cut in G can be obtained by solving a global edge capacitated cut in a directed graph using standard reductions between flow/cut variants. By using the result of Hao et al.[27], it can be done in time $O(n^3 \log n)$. Kratsch et al.[30, 31] presented an algorithm parameterized by the modular width mw that solves the global minimum vertex cut problem in time $O(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$. We present here an algorithm parameterized by the neighborhood diversity that solves the problem in time $O(\mathsf{nd}^3 \log \mathsf{nd} + n + m)$.

Algorithm. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_t\}$ be the type partition of G and let $H = (V(H), E(H))$ be the type graph. Consider the capacity function $c_H : V(H) \to R^+$ of the type graph H defined as $c_H(i) = \sum_{u \in V_i} c(u)$. The algorithm first finds the global minimum vertex cut X_H of H with capacities c_H, and then returns $X = \bigcup_{i \in X_H} V_i$.

We will prove that the set X returned by the algorithm is a global minimum vertex cut of G with capacity c. To this aim, we first characterize a global minimum vertex cut in terms of the type partition of G.

▶ **Lemma 7.** *Let $G = (V, E)$ be a graph with type partition $V_1, V_2, \ldots, V_t$. Let $c : V \to R^+$ be the capacity function of G and let X be any global minimum vertex cut of G with capacity c.*

a) *For each disconnected partition A_X, B_X of $G - X$, any vertex $u \in X$ must have at least a neighbor in A_X and at least a neighbor in B_X.*

b) *For each $i = 1, \ldots, t$, either $V_i \subseteq X$ or $V_i \cap X = \emptyset$.*

c) *There exists a disconnected partition A_X, B_X of $G - X$ such that either $V_i \subseteq A_X$ or $V_i \cap A_X = \emptyset$ (resp. either $V_i \subseteq B_X$ or $V_i \cap B_X = \emptyset$), $i = 1, \ldots, t$.*

Proof. To prove a) we proceed by contradiction and assume that there exists a vertex $u \in X$ that has only neighbors in $X \cup A_X$ (resp. in $X \cup B_X$). In this case $X - \{u\}$ is a global vertex cut whose capacity is less than that of X, and this contradicts the minimum capacity of X.

The proof of b) is again by contradiction. Suppose that there exists a vertex $u \in X \cap V_i$ and a vertex $v \in V_i - X$. W.l.o.g., suppose that $v \in A_X$. Since $u, v \in V_i$, they share the same neighborhood and by a) vertex u must have at least a neighbor in B_X. Hence, also v must have at least a neighbor in B_X, thus contradicting the assumption that $G - X$ is disconnected.

Finally, we prove c). Let A_X, B_X be any disconnected partition of $G - X$ and let $u \in A_X \cap V_i$ (and $V_i \not\subseteq A_X$). By b) no vertex in V_i is in X. We have only to consider the possibility that $B_X \cap V_i \neq \emptyset$. This is not possible if V_i induces a clique in G since otherwise A_X and B_X should be connected by at least one edge. In case V_i induces an independent set in G, we can move each vertex in $B_X \cap V_i$ from B_X to A_X obtaining again a disconnected partition of $G - X$. Hence, $A_X \cup (B_X \cap V_i)$ and $B_X - V_i$ is a disconnected partition of $G - X$. This proves that it is possible to find a disconnected partition of $G - X$ satisfying c). ◀

▶ **Theorem 8.** *The set X returned by the algorithm is a global minimum vertex cut with capacity c.*

Proof. We first prove that X is a vertex cut of G, that is we prove that $G - X$ is disconnected. By the fact that X_H is a global minimum vertex cut of H with capacity c_H, we can find a disconnected partition A_H, B_H of $H - X_H$ and so we have that each $i \in A_H$ has all its neighbors in $A_H \cup X_H$ and, each $j \in B_H$ has all its neighbors in $B_H \cup X_H$. By the construction of X in the algorithm, i.e., $X = \bigcup_{i \in X_H} V_i$, we can pinpoint the sets $A = \bigcup_{i \in A_H} V_i$ and $B = \bigcup_{i \in B_H} V_i$, that are a partition of $V(G) - X$. Now, we prove that there is no edge between vertices of A and B. Fix $u \in A$. Since $u \in V_i$ for some $i \in A_H$ and using the fact that A_H and B_H are a disconnected partition of $H - X_H$ and so u cannot be neighbor of any vertex in same set V_j for $j \in B_H$, we have that vertex u has all its neighbors in $A \cup X$. In the same way we can prove that each $v \in B$ has all its neighbors in $B \cup X$. This proves that A, B is a disconnected partition of $G - X$.

Finally, we prove that X has minimum capacity, so completing the proof. By contradiction suppose there exists a global minimum vertex cut Y with $c(Y) < c(X)$. By b) in Lemma 7 we can define the set $Y_H = \{i \in V(H) \mid V_i \subseteq Y\}$. Let A_Y and B_Y be the disconnected partition of $G - Y$ for which c) in Lemma 7 holds. Hence, if we define $A = \{i \in V(H) \mid V_i \subseteq A_Y\}$ and $B = \{j \in V(H) \mid V_j \subseteq B_Y\}$, then each $i \in A$ has all its neighbors in $A \cup Y_H$, and each $j \in B$ has all its neighbors in $B \cup Y_H$ proving that Y_H is a vertex cut of H. Furthermore,

$$\begin{aligned} c(Y_H) &= \sum_{i \in Y_H} c_H(i) = \sum_{i \in Y_H} \sum_{u \in V_i} c(u) = \sum_{u \in Y} c(u) = c(Y) \\ &< c(X) = \sum_{u \in X} c(u) = \sum_{i \in X_H} \sum_{u \in V_i} c(u) = \sum_{i \in X_H} c_H(i) \\ &= c(X_H), \end{aligned}$$

which contradicts the assumption that X_H is a global minimum vertex cut of H respect to the capacity function c_H. ◀

Running time. The running time of our algorithm strongly depends on the time to compute the global minimum vertex cut X_H of H with capacities c_H. By using the best known algorithm to evaluate the global minimum vertex cut, we have that X_H can be computed in time $O(\mathsf{nd}^3 \log \mathsf{nd})$.

▶ **Theorem 9.** *For any graph $G = (V, E)$ and capacity function $c : V \to R^+$, the* GLOBAL MINIMUM VERTEX CUT *problem can be solved in time $O(\mathsf{nd}^3 \log \mathsf{nd} + n + m)$.*

References

1 A. Abboud, F. Grandoni, and V. V. Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *ACM-SIAM Symposium on Discrete Algorithms (SODA), 1681–1697*, 2015.

2 A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *IEEE Symposium on Foundations of Computer Science (FOCS'14). IEEE, 434–443*, 2014.

3 F. N. Abu-Khzam, S. Li, C. Markarian, F. Meyer auf der Heide, and P. Podlipyan. Modular-Width: An Auxiliary Parameter for Parameterized Parallel Complexity. In *Frontiers in Algorithmics, FAW 2017, LNCS 10336*, 2017.

4 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica, 17(3): 209–223*, 1997.

5 S. E. Asch. Studies of independence and conformity: A minority of one against a unanimous majority. *Psychological Monographs, 70*, 1956.

6 L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data 4 (3)*, 2010.

7 R. Belmonte, F. V. Fomin, P. A. Golovach, and M. S. Ramanujan. Metric Dimension of Bounded Width Graphs. *Mathematical Foundations of Computer Science (MFCS '15), LNCS 923*, 2015.

8 M. Bentert, T. Fluschnik, A. Nichterlein, and R. Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences, 103, 61–77*, 2019.

9 M. Borassi, P. Crescenzi, and M. Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science, 322:51–67*, 2016.

10 G. Cordasco, L. Gargano, A. A. Rescigno, and U. Vaccaro. Optimizing spread of influence in social networks via partial incentives. In *22nd International Colloquium on Structural Information and Communication Complexity, SIROCCO 2015. LNCS 9439, 119–134*, 2015.

11 G. Cordasco, L. Gargano, A. A. Rescigno, and U. Vaccaro. Evangelism in social networks: Algorithms and complexity. *Networks 71(4): 346–357*, 2018.

12 D. Coudert, G. Ducoffe, and A. Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18), 2765–2784*, 2018.

13 M. Doucha and J. Kratochvíl. Cluster Vertex Deletion: A Parameterization between Vertex Cover and Clique-Width. *MFCS 2012, 348–359*, 2012.

14 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 2012.

15 P. Dvorák, D. Knop, and T. Toufar. Target Set Selection in Dense Graph Classes. In *arXiv:1610.07530*, 2016.

16 D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, ISBN:0521195330, 2010.

17 J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics, 17(3):449–467*, 1965.

18 J. Fiala, T. Gavenciak, D. Knop, M. Koutecky, and J. Kratochvíl. Fixed parameter complexity of distance constrained labeling and uniform channel assignment problems. In *arXiv:1507.00640*, 2015.

19 F. V. Fomin, M. Liedloff, P. Montealegre, and I. Todinca. Algorithms Parameterized by Vertex Cover and Modular Width, through Potential Maximal Cliques. In *Ravi R., Gørtz I.L. (eds) Algorithm Theory – SWAT 2014, LNCS vol 8503, Springer*, 2014.

20 H. N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms, Vol. 14 (3), Article 39*, 2018.

21 J. Gajarský, M. Lampis, and S. Ordyniak. Parameterized Algorithms for Modular-Width. In *Gutin G., Szeider S. (eds) Parameterized and Exact Computation, IPEC 2013, LNCS 8246*, 2013.

22 A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry, 5(3):165–185*, 1995.

23 F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic ComputationJuly, ISSAC '14, 296–303*, 2014.

24 R. Ganian. Using neighborhood diversity to solve hard problems. In *arXiv:1201.3091*, 2012.

25 L. Gargano and A.A. Rescigno. Complexity of conflict-free colorings of graphs. *Theoretical Computer Science, 566, 39–49*, 2015.

26 A. C. Giannopoulou, G. B. Mertzios, and R. Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science, 689: 67–95*, 2017.

27 J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms, 17(3):424–446*, 1994.

28 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing, 7(4):413–423*, 1978.

29 M. N. Kolountzakis, G. L. Miller, R. Peng, and C. E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Math. 8 (1–2), 161–185*, 2012.

30 D. Kratsch and J. P. Spinrad. Between $o(nm)$ and $o(n^\alpha)$. *SIAM Journal on Computing, 36(2):310–325*, 2006.

31 S. Kratsch and F. Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In *arXiv:1804.10173*, 2018.

32 M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica 64, 19–37*, 2012.

33 G. B. Mertzios, A. Nichterlein, , and R. Niedermeier. The power of linear-time data reduction for maximum matching. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science (MFCS'17), 83, 46:1–46:14*, 2017.

34 S. Micali and V. V. Vazirani. An $o(sqrt(|V|)|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, 17–27*, 1980.

35 M. E. J. Newman. The structure and function of complex networks. *SIAM Rev. 45 (2), 167–256*, 2003.

36 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

37 V. V. Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *International Symposium on Parameterized and Exact Computation (IPEC), 16–28*, 2015.

38 V. V. Williams and R. R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *IEEE Symposium on Foundations of Computer Science (FOCS), 645–654*, 2010.

39 Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *Proceedings of the 28th International Conference on Data Engineering, ICDE '12, IEEE Computer Society, 1049–1060*, 2012.

Physical Zero-Knowledge Proof for Numberlink

Suthee Ruangwises
Department of Mathematical and Computing Science, Tokyo Institute of Technology, Japan
ruangwises.s.aa@m.titech.ac.jp

Toshiya Itoh
Department of Mathematical and Computing Science, Tokyo Institute of Technology, Japan
titoh@c.titech.ac.jp

Abstract

Numberlink is a logic puzzle for which the player has to connect all pairs of cells with the same numbers by non-crossing paths in a rectangular grid. In this paper, we propose a physical protocol of zero-knowledge proof for Numberlink using a deck of cards, which allows a player to physically show that he/she knows a solution without revealing it. In particular, we develop a physical protocol to count the number of elements in a list that are equal to a given secret value without revealing that value, the positions of elements in the list that are equal to it, or the value of any other element in the list. Our protocol can also be applied to verify the existence of vertex-disjoint paths connecting all given pairs of endpoints in any undirected graph.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques; Theory of computation → Cryptographic protocols

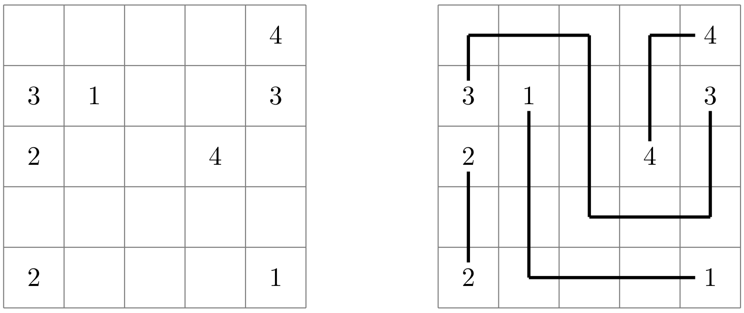
Keywords and phrases Zero-knowledge proof, Card-based cryptography, Numberlink, Puzzles, Games

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.22

1 Introduction

Numberlink is a logic puzzle introduced by a Japanese company Nikoli famous for developing many popular puzzles including Sudoku, Akari, Makaro, and Norinori. The puzzle has become increasingly popular and a large number of Numberlink mobile apps with different names and slightly different variants of rule have been developed [8].

A Numberlink puzzle consists of a rectangular grid with some cells containing a number. Each number appears exactly twice in the grid. The goal of this puzzle is to connect every pair of the same numbers by a path that can go from a cell to its horizontally or vertically adjacent cell. Paths cannot cross or share a cell with one another. In the official rule [16], it is not required that all cells in the grid have to be covered by paths. However, a puzzle is generally considered to be well-designed if it has a unique solution, and all cells are covered by paths in that solution.



				4
3	1			3
2			4	
2				1

Figure 1 An example of a Numberlink puzzle (left) and its solution (right).

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 22; pp. 22:1–22:11
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Suppose that Anna, an expert in Numberlink, created a difficult Numberlink puzzle and challenged her friend Brian to solve it. Unluckily, after several tries Brian could not solve her puzzle. He then claimed that the puzzle has no solution and refused to try it anymore. How can Anna convince bad-luck Brian that her puzzle actually has a solution without revealing it to him (which would render the challenge pointless)?

1.1 Zero-Knowledge Proof

A zero-knowledge proof is an interactive proof between a prover P and a verifier V, with both given an instance x of a computational problem. Only P knows a solution w of x, and the computational power of V is limited so that he/she cannot obtain w from x. P wants to convince V that he/she knows w without revealing any information about w to V. A zero-knowledge proof must satisfies the following three properties.

1. **Completeness:** If P knows w, then P is able to convince V with high probability (in this paper, we are interested in the *perfect completeness* property where the probability to convince V is one).
2. **Soundness:** If P does not know w, then P is not able to convince V, except with a small probability called *soundness error* (in this paper, we are interested in the *perfect soundness* property where the soundness error is zero).
3. **Zero-Knowledge:** V cannot obtain any information about w, i.e. there exists a probabilistic polynomial time algorithm S (called the simulator) that does not know w, and the outputs of S follow the same probability distribution as the outputs of the real protocol.

The concept of zero-knowledge proof was first introduced by Goldwasser et al. [7], and it was proved by Goldreich et al. [6] that a zero-knowledge proof exists for any NP problem. Because Numberlink has been proved to be NP-complete [1, 13, 14], it is possible to construct a cryptographic zero-knowledge proof for Numberlink. However, such construction requires cryptographic primitives and is not intuitive or practical.

Instead, we are interested in constructing a physical protocol of zero-knowledge proof using a deck of playing cards. The benefit of such protocols is that they use only a small deck of cards which can be found in everyday life and do not require computers. Moreover, these intuitive protocols are easy to understand and verify the security and correctness, even for non-experts, and thus can be used as examples for didactic purpose.

1.2 Related Work

In 2007, Gradwohl et al. [9] developed the first physical protocols of zero-knowledge proof for Sudoku. Each of their several variants of the protocol either uses special scratch-off cards or has a non-zero soundness error. Later, Sasaki et al. [17] improved the protocol for Sudoku to achieve perfect soundness without using special cards. Besides Sudoku, physical protocols of zero-knowledge proof for other logic puzzles have been developed as well, including Nonogram [4], Akari [2], Takuzu [2], Kakuro [2, 15], KenKen [2], Makaro [3], and Norinori [5].

These protocols of zero-knowledge proof employ methods to physically verify specific functions. For example, the protocol for Sudoku [9] shows how to verify the presence of all numbers in a list without revealing their order, the protocol for Makaro [3] shows how to verify that a number is the largest one in a list without revealing any value in the list, and the protocol for Norinori [5] shows how to verify the presence of a given number in a list without revealing its position or any other value in the list.

1.3 Our Contribution

In this paper, we propose a physical protocol of zero-knowledge proof with perfect completeness and perfect soundness for Numberlink using a deck of cards. More importantly, we also extend the set of functions that are known to be physically verifiable.

By developing the protocol for Numberlink, we show in particular how to count the number of elements in a list that are equal to a given secret value without revealing that value, the positions of elements in the list that are equal to it, or the value of any other element in the list. Also, by transforming a graph problem into this element-counting problem, we show how to verify the existence of vertex-disjoint paths connecting all given pairs of endpoints in any undirected graph.

2 Preliminaries

2.1 Numberlink Board

Suppose that a Numberlink grid has size $m \times n$, and has k pairs of numbers $1, 2, ..., k$ written on it. We call two cells in the grid *adjacent* if they are horizontally or vertically adjacent. Cells with a number written on them are called *terminal cells*; other cells are called *non-terminal cells*.

A *path* in a valid solution of a Numberlink puzzle is a sequence of cells $(c_1, c_2, ..., c_t)$ where c_1 and c_t are terminal cells with the same numbers written on them and all other cells are non-terminal cells, with c_i being adjacent to c_{i+1} for every $i = 1, 2, ..., t-1$. Also, a path $(c_1, c_2, ..., c_t)$ is called *simple* if there is no i, j such that $j > i+1$ and c_i is adjacent to c_j.

A Numberlink puzzle is called *well-designed* if it has a unique solution, and all cells are covered by paths in that solution. Observe that if a puzzle is well-designed, then every path in its solution must be simple (otherwise if we have a non-simple path $(c_1, c_2, ..., c_t)$ with c_i being adjacent to c_j where $j > i+1$, then we can replace it with a shorter path $(c_1, c_2, ...c_i, c_j, c_{j+1}..., c_t)$, thus creating an alternative solution).

2.2 Cards

In our protocol, we use two types of cards: *encoding cards* and *marking cards*. An encoding card has either ♣ or ♡ on the front side, while a marking card has a positive integer on the front side. All cards have an identical back side.

Define $E_y(x)$ to be a sequence of y encoding cards, with all cards being $\boxed{\clubsuit}$ except the x-th card from the left being $\boxed{\heartsuit}$. For example, $E_3(1)$ is $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\clubsuit}$ and $E_4(3)$ is $\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}$. We use $E_y(x)$ to encode a number x in the situation where the maximum possible number is at most y.

2.3 Matrix

Suppose we have a numbers $x_1, x_2, ..., x_a$, with each of them being at most b. Each number x_i is encoded by a sequence of cards $E_b(x_i)$. We construct a matrix $D(a, b)$ of cards by the following procedures.

First, create an $a \times b$ matrix of face-down encoding cards, with the i-th topmost row being $E_b(x_i)$. Then, on top of the topmost row of the matrix, place face-down marking cards $1, 2, ..., b$ from left to right in this order. We call this new row Row 0. Also, to the left of the leftmost column of the matrix, place face-down marking cards $2, 3, ..., a$ from top to bottom

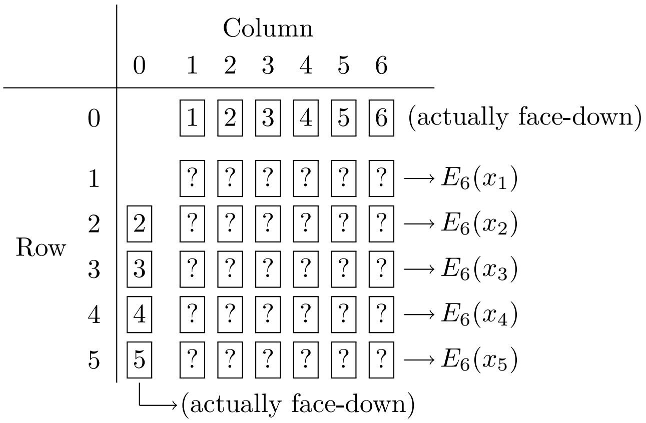

Column
0 1 2 3 4 5 6
Row
0
1
2
3
4
5
1 2 3 4 5 6
(actually face-down)
?
2
3
4
5
$\longrightarrow E_6(x_1)$
$\longrightarrow E_6(x_2)$
$\longrightarrow E_6(x_3)$
$\longrightarrow E_6(x_4)$
$\longrightarrow E_6(x_5)$
(actually face-down)

Figure 2 An example of a matrix $D(5, 6)$.

in this order (starting at Row 2). We call this new column Column 0. As a result, $D(a, b)$ becomes an incomplete $(a+1) \times (b+1)$ matrix with two cards at the top-left corner removed (see Figure 2).

We will then introduce the operations that will be applied to the matrix $D(a, b)$.

2.4 Double-Scramble Shuffle

A *double-scramble shuffle* is an extension of a *pile-scramble shuffle* first developed by Ishikawa et al. [12]. In the pile-scramble shuffle, we shuffle only the columns of the matrix by a random permutation; in the double-scramble shuffle, we shuffle both the selected rows and selected columns of the matrix by random permutations.

The formal procedures of the double-scramble shuffle are as follows.

1. Uniformly select a permutation $p = (p_2, p_3, ..., p_a)$ of $(2, 3, ..., a)$ at random. p must be unknown to the verifier.
2. Secretly rearrange Rows $2, 3, ..., a$ by a permutation p, i.e. move Row i to Row p_i for every $i = 2, 3, ..., a$.
3. Uniformly select a permutation $q = (q_1, q_2, ..., q_b)$ of $(1, 2, ..., b)$ at random. q must be unknown to the verifier.
4. Secretly rearrange Columns $1, 2, ..., b$ by a permutation q, i.e. move Column j to Column q_j for every $j = 1, 2, ..., b$.

Observe that the double-scramble shuffle makes the order of $x_2, x_3, ..., x_a$ indifferent to the verifier. It also hides the actual value of each x_i, but preserves the number of rows that encode the same value as Row 1.

▶ Remark 1. In real world, the prover P can perform the double-scramble shuffle by the following procedures. In Step 2, P publicly puts the cards in each row into an envelope and seal it. Then, P rearranges the envelopes by a permutation p without the verifier V observing. Finally, P publicly opens each envelope and put the cards in it back into a corresponding row. By doing this, V can ensure that P has only made row-wise swaps (and not arbitrary exchanges of cards) without knowing the permutation p. The same goes for column-wise swaps in Step 4.

2.5 Rearrangement Protocol

A rearrangement protocol was implicitly used in some previous work on card-based protocols [3, 10, 11, 17]. The sole purpose of this protocol is to revert the cards (after we perform some operations on them) back to their original positions so that we can reuse the cards without revealing them.

The formal procedures of the rearrangement protocol are as follows.

1. Apply the double-scramble shuffle to the matrix.
2. Publicly turn over all marking cards in Column 0. Suppose the opened cards are $p_2, p_3, ..., p_a$ from top to bottom in this order.
3. Publicly rearrange Rows $2, 3, ..., a$ by a permutation $p = (p_2, p_3, ..., p_a)$, i.e. move Row i to Row p_i for every $i = 2, 3, ..., a$.
4. Publicly turn over all marking cards in Row 0. Suppose the opened cards are $q_1, q_2, ..., q_b$ from left to right in this order.
5. Publicly rearrange Columns $1, 2, ..., b$ by a permutation $q = (q_1, q_2, ..., q_b)$, i.e. move Column j to Column q_j for every $j = 1, 2, ..., b$.

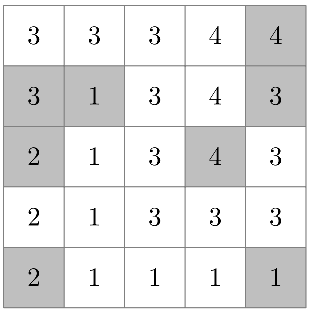
Note that since we first apply the double-scramble shuffle at Step 1, the order of Rows $2, 3, ..., a$ and the order of Columns $1, 2, ..., b$ are uniformly distributed among all possible permutations. Therefore, revealing marking cards in Steps 2 and 4 does not leak any information about the cards.

3 Our Main Protocol

3.1 Well-Designed Puzzles

For simplicity, we first consider a special case where the puzzle is well-designed.

Recall that a Numberlink grid has size $m \times n$, and has k pairs of numbers $1, 2, ..., k$ written on it. In the solution, for each path joining two terminal cells with a number x, we write a number x on every cell that the path passes through (see Figure 3). Since this is a well-designed puzzle, every path is simple and every cell has a number on it.



3	3	3	4	4
3	1	3	4	3
2	1	3	4	3
2	1	3	3	3
2	1	1	1	1

Figure 3 Transformation of the solution of the puzzle in Figure 1, with gray cells being terminal cells.

The intuition of our protocol is that the prover P will try to convince the verifier V that

1. every terminal cell has exactly one adjacent cell with the same number, and
2. every non-terminal cell has exactly two adjacent cells with the same number.

First, for each terminal cell with a number x, P publicly puts a sequence of cards $E_k(x)$ on it. Then, for each non-terminal cell with a number x, P secretly puts a sequence of cards $E_k(x)$ on it.

The verification phase for each terminal cell c works as follows.

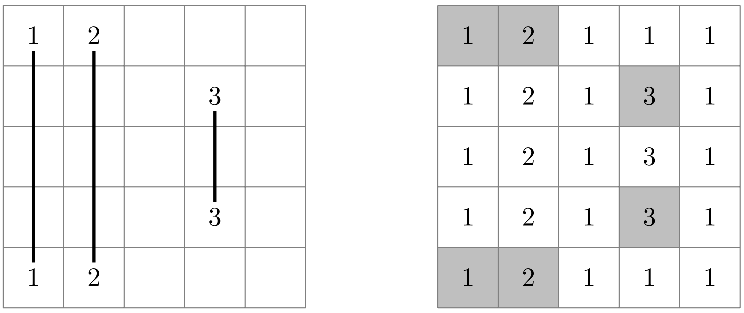
1. Publicly construct a matrix of cards in the following way: put the sequence on c into Row 1, then put the sequence on each adjacent cell to c in any order into each of the next four (or three, or two, if c is on the edge or at the corner) rows. Finally, put the number cards to complete the matrix $D(5, k)$ (or $D(4, k)$, or $D(3, k)$, for the edge or corner case).
2. Apply the double-scramble shuffle to the matrix.
3. Publicly turn over all encoding cards in Row 1. Locate the position of a $\boxed{\heartsuit}$. Suppose it is at Column j.
4. Publicly turn over all other encoding cards in Column j. If there is exactly one $\boxed{\heartsuit}$ besides the one in Row 1, then the protocol continues; otherwise V rejects and the protocol terminates.
5. Apply the rearrangement protocol to the matrix to revert the cards to their original positions, and publicly put the cards back to their corresponding cells.

The verification phase for each non-terminal cell works exactly the same as that for a terminal cell, except that in Step 4, V has to verify that there are exactly two (instead of one) $\boxed{\heartsuit}$s in Column j besides the one in Row 1.

P performs the verification phase for every cell in the grid. If every cell passes the verification, then V accepts.

In total, our protocol in the setting of a well-designed puzzle uses kmn encoding cards and $k + 4$ marking cards.

▶ Remark 2. Note that in this protocol, P can convince V that the solution he/she has is valid, but cannot convince V that the puzzle is well-designed or that all cells are covered by paths in his/her solution (see Figure 4).



1	2	1	1	1
1	2	1	3	1
1	2	1	3	1
1	2	1	3	1
1	2	1	1	1

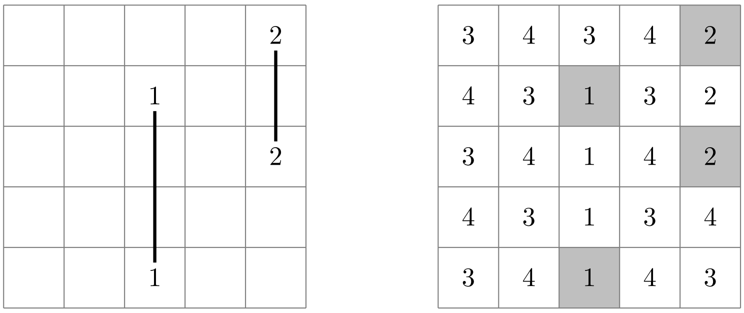
Figure 4 In a puzzle that is not well-designed, the prover P knows a solution that does not cover all cells (left), but it is possible for P to run this protocol with a non-solution (right) and get accepted.

3.2 General Puzzles

Now we consider a general case where the puzzle may not be well-designed, and the paths in our solution may not cover all cells. We can still apply a protocol similar to the one for well-designed puzzles, but with some additional tricks employed.

First, if our solution contains a non-simple path $(c_1, c_2, ..., c_t)$ with c_i being adjacent to c_j where $j > i + 1$, then we replace it with a shorter path $(c_1, c_2, ...c_i, c_j, c_{j+1} ..., c_t)$. We repeatedly perform this until every path in the solution becomes simple.

We put a number on each cell that is covered by a path the same way as in the well-designed setting. Then, for each cell in the i-th row and j-th column that is not covered by any path, we put a number $k+1$ on it if $i+j$ is even and put a number $k+2$ on it if $i+j$ is odd (see Figure 5). Note that by filling the numbers this way, each cell not covered by any path will have no adjacent cell with the same number.



3	4	3	4	2
4	3	1	3	2
3	4	1	4	2
4	3	1	3	4
3	4	1	4	3

Figure 5 An example of a solution of a puzzle that is not well-designed (left), and the way we put numbers on the grid (right).

The intuition of our protocol in this setting is that the prover P will try to convince the verifier V that

1. every terminal cell has exactly one adjacent cell with the same number, and
2. every non-terminal cell either has a number $k+1$ or $k+2$, or has exactly two adjacent cells with the same number.

Since the maximum number on the grid is at most $k+2$ in this setting, we always use $E_{k+2}(x)$ instead of $E_k(x)$ to encode a number x. For each terminal cell with a number x, P publicly puts a sequence of cards $E_{k+2}(x)$ on it. Then, for each non-terminal cell with a number x, P secretly puts a sequence of cards $E_{k+2}(x)$ on it.

For each terminal cell, the verification phase works exactly the same as in the well-designed setting (except the width of the matrix will be $k+2$ instead of k). For each non-terminal cell, we add four additional rows, two encoding the number $k+1$ and two encoding the number $k+2$, to the bottom of the matrix. The formal steps for verifying each non-terminal cell c are as follows.

1. Publicly construct a matrix of cards in the following way: put the sequence on c into Row 1, then put the sequence on each adjacent cell to c into each of the next four (or three, or two, if c is on the edge or at the corner) rows in any order. Then, put the sequences $E_{k+2}(k+1)$, $E_{k+2}(k+1)$, $E_{k+2}(k+2)$, and $E_{k+2}(k+2)$ into each of the next four rows in any order. Finally, put the number cards to complete the matrix $D(9,k+2)$ (or $D(8,k+2)$, or $D(7,k+2)$, for the edge or corner case).
2. Apply the double-scramble shuffle to the matrix.
3. Publicly turn over all encoding cards in Row 1. Locate the position of a $\boxed{\heartsuit}$. Suppose it is at Column j.
4. Publicly turn over all other encoding cards in Column j. If there are exactly two $\boxed{\heartsuit}$s besides the one in Row 1, then the protocol continues; otherwise V rejects and the protocol terminates.
5. Apply the rearrangement protocol to the matrix to revert the cards to their original positions, and publicly put the cards back to their corresponding cells.

In total, our protocol in the setting of a general puzzle uses $(k+2)(mn+4)$ encoding cards and $k+10$ marking cards.

4 Proof of Security

We will prove the perfect completeness, perfect soundness, and zero-knowledge properties of our protocol. We will consider only the protocol in a general setting as it can be used for all Numberlink instances.

▶ **Lemma 3** (Perfect Completeness). *If P knows a solution of the Numberlink puzzle, then V always accepts.*

Proof. Suppose that P knows a solution that contains only simple paths, and fills numbers on the grid according to that solution.

- Consider each terminal cell c with a number $x \leq k$. There must be a path $(c_1, c_2, ..., c_t)$ starting at $c_1 = c$ and ending at c_t, the other terminal cell with the same number x. Since each cell in the grid either belongs to some path or has a number $k+1$ or $k+2$ on it, the set of all cells having a number x is exactly $\{c_1, c_2, ..., c_t\}$. We know that c_2 is adjacent to c and has a number x on it. Moreover, since the path is simple, there cannot be an index $i > 2$ such that c_i is adjacent to c. Therefore, c has exactly one adjacent cell with the same number. Since the double-scramble shuffle preserves the number of rows that encode a value equal to that of Row 1, the verification phase for c will pass.
- Consider each non-terminal cell c with a number $x \leq k$. There must be a path $(c_1, c_2, ..., c_t)$ joining two terminal cells with a number x. As previously shown, the set of all cells having a number x is exactly $\{c_1, c_2, ..., c_t\}$, so we have $c = c_i$ for some index i where $1 < i < t$. We know that c_{i-1} and c_{i+1} are adjacent to c and have a number x on them. Moreover, since the path is simple, there cannot be an index j other than $i-1$ and $i+1$ such that c_j is adjacent to c. Therefore, c has exactly two adjacent cells with the same number. Since the double-scramble shuffle preserves the number of rows that encode a value equal to that of Row 1, the verification phase for c will pass.
- Consider each non-terminal cell c with a number $x = k+1$ or $k+2$. Recall that by the way we put numbers on the cells not covered by any path, c has no adjacent cell with the same number. However, in the verification phase of c, we add four additional rows, two encoding $k+1$ and two encoding $k+2$, to the matrix. Therefore, there will be two rows that encode a value equal to that of Row 1, hence the verification phase for c will pass. ◀

▶ **Lemma 4** (Perfect Soundness). *If P does not know a solution of the Numberlink puzzle, then V always rejects.*

Proof. We will prove the contrapositive of this statement. Suppose that V accepts, meaning that the verification phase passes for every cell.

Consider each number $x \leq k$. We know that there are two terminal cells with the number x. Consider one of them, called c_1. We know from the verification phase that c_1 has exactly one adjacent cell, called c_2, with a number x. For each $i \geq 2$, if c_i is a terminal cell, then there exists a path $(c_1, c_2, ..., c_i)$ connecting the two terminal cells with the number x. Otherwise if c_i is a non-terminal cell, then we know from the verification phase that c_i has exactly two adjacent cells with the number x, one of them being c_{i-1}. We then inductively move to consider the other cell, called c_{i+1}, in the same manner. Since the path is simple, c_{i+1} must be different from any c_j with $j \leq i$. Therefore, we must eventually reach the other terminal cell, implying that there exists a path connecting the two terminal cells with the number x. Since this is true for every number $x \leq k$, there exists a path joining every pair of terminal cells with the same numbers in P's solution, which means P must know a valid solution. ◀

▶ **Lemma 5** (Zero-Knowledge). *During the verification phase, V learns nothing about P's solution of the Numberlink puzzle.*

Proof. To prove the zero-knowledge property, it is sufficient to prove that all distributions of the values that appear when we turn over cards can be simulated without knowing P's solution.

Consider the verification phase of a cell c with a matrix $D(a, k+2)$ of cards ($a \in \{3, 4, 5\}$ for a terminal cell and $a \in \{7, 8, 9\}$ for a non-terminal cell). There are two steps in the verification phase where we turn over cards.

In the step where we turn over all encoding cards in Row 1, the order of Columns $1, 2, ..., k+2$ is uniformly distributed among all possible permutations due to the double-scramble shuffle, hence the $\boxed{\heartsuit}$ has an equal probability to appear at each of the $k+2$ positions. Therefore, this step can be simulated without knowing the solution.

After that, we locate the position of the $\boxed{\heartsuit}$ in Row 1 to be at Column j, and then turn over all other encoding cards in Column j. The order of Rows $2, 3, ..., a$ is uniformly distributed among all possible permutations due to the double-scramble shuffle, hence all (one or two) $\boxed{\heartsuit}$s have an equal probability to appear at each of the $a-1$ positions. Therefore, this step can be simulated without knowing the solution. ◀

5 Applications

Besides the Numberlink puzzle, our technique of transforming a graph problem into an element-counting problem can be applied to verify the existence of vertex-disjoint paths connecting k given pairs of endpoints in any undirected graph G.

In this setting, a path $(v_1, v_2, ..., v_t)$ is called *simple* if there is no i, j such that $j > i+1$ and v_i is adjacent to v_j. Similarly to the original protocol for Numberlink, if our solution contains a non-simple path $(v_1, v_2, ..., v_t)$ with v_i being adjacent to v_j where $j > i+1$, then we replace it with a shorter path $(v_1, v_2, ...v_i, v_j, v_{j+1}..., v_t)$. We repeatedly perform this until every path in the solution becomes simple.

Suppose that the maximum degree of a vertex in G is d. We can inductively color the vertices of G with at most $d+1$ colors in linear time such that no adjacent vertices have the same color. Similarly to the protocol for Numberlink, for each path connecting the x-th given pair of endpoints, we put a number x on every vertex on that path. For each vertex v not covered by any path, we put a number $k+i$ on v if it has the i-th color in our $(d+1)$-coloring of G. By putting the numbers this way, each vertex not covered by any path will have no neighbor with the same number.

Let T be the set of vertices that appear in the list of k given pairs of endpoints. The intuition of our protocol is that the prover P will try to convince the verifier V that

1. every vertex in T has exactly one adjacent vertex with the same number, and
2. every vertex not in T either has a number greater than k, or has exactly two adjacent vertices with the same number.

Since the maximum number on the vertices is at most $k+d+1$, we use $E_{k+d+1}(x)$ to encode a number x. For each vertex $v \in T$ with a number x, P publicly puts a sequence of cards $E_{k+d+1}(x)$ on v. Then, for each vertex $v \notin T$ with a number x, P secretly puts a sequence of cards $E_{k+d+1}(x)$ on v.

The verification phase works in the same manner as in the protocol for Numberlink in a general setting. For a vertex $v \in T$, P puts the sequence on v into the first row, and the sequence on each of v's neighbors into each of the next (at most) d rows of the matrix. The verifier V has to verify that there is exactly one $\boxed{\heartsuit}$ in the same column as the $\boxed{\heartsuit}$ in Row 1. For a vertex $v \notin T$, P does the same but also puts $2(d+1)$ additional rows to the matrix, with two of them encoding each of the numbers $k+1, k+2, ..., k+d+1$. V then has to verify that there are exactly two $\boxed{\heartsuit}$s in the same column as the $\boxed{\heartsuit}$ in Row 1.

In total, this protocol uses $(k+d+1)(|V_G|+2d+2)$ encoding cards and $k+4d+3$ marking cards, where V_G is the set of vertices of G.

6 Future Work

We developed a physical protocol of zero-knowledge proof for Numberlink using $\Theta(kmn)$ cards. A challenging future work involving Numberlink is to develop a protocol of zero-knowledge proof that requires asymptotically fewer number of cards, or the one that can convince the verifier that the prover's solution contains paths that cover all cells (which is apparently a requirement in a variant of rule used in some newly developed mobile apps).

Other possible future work includes developing protocols of zero-knowledge proof for other logic puzzles, as well as exploring the method to physically verify other interesting functions.

References

1 A. Adcock, E.D. Demaine, M.L. Demaine, M.P. O'Brien, F. Reidl, F.S. Villaamil, and B.D. Sullivan. Zig-Zag Numberlink is NP-complete. *Journal of Information Processing*, 23(3):239–245, 2015. doi:10.2197/ipsjjip.23.239.

2 X. Bultel, J. Dreier, J.-G. Dumas, and P. Lafourcade. Physical zero-knowledge proofs for Akari, Takuzu, Kakuro and KenKen. In *Proceedings of the 8th International Conference on Fun with Algorithms (FUN)*, pages 8:1–8:20, 2016. doi:10.4230/LIPIcs.FUN.2016.8.

3 X. Bultel, J. Dreier, J.-G. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, A. Nagao, T. Sasaki, K. Shinagawa, and H. Sone. Physical zero-knowledge proof for Makaro. In *Proceedings of the 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 111–125, 2018. doi:10.1007/978-3-030-03232-6_8.

4 Y.-F. Chien and W.-K. Hon. Cryptographic and physical zero-knowledge proof: From Sudoku to Nonogram. In *Proceedings of the 5th International Conference on Fun with Algorithms (FUN)*, pages 102–112, 2010. doi:10.1007/978-3-642-13122-6_12.

5 J.-G. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, T. Sasaki, and H. Sone. Interactive physical zero-knowledge proof for Norinori. In *Proceedings of the 25th International Computing and Combinatorics Conference (COCOON)*, pages 166–177, 2019. doi:10.1007/978-3-030-26176-4_14.

6 O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *Journal of the ACM*, 38(3):691–729, 1991. doi:10.1145/116825.116852.

7 S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. doi:10.1137/0218012.

8 Google Play: Numberlink. https://play.google.com/store/search?q=Numberlink.

9 R. Gradwohl, M. Naor, B. Pinkas, and G.N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of Sudoku puzzles. In *Proceedings of the 4th International Conference on Fun with Algorithms (FUN)*, pages 166–182, 2007. doi:10.1007/978-3-540-72914-3_16.

10 Y. Hashimoto, K. Shinagawa, K. Nuida, M. Inamura, and G. Hanaoka. Secure grouping protocol using a deck of cards. In *Proceedings of the 10th International Conference on Information Theoretic Security (ICITS)*, pages 135–152, 2017. doi:10.1007/978-3-319-72089-0_8.

11 T. Ibaraki and Y. Manabe. A more efficient card-based protocol for generating a random permutation without fixed points. In *Proceedings of the 3rd International Conference on Mathematics and Computers in Sciences and Industry (MCSI)*, pages 252–257, 2016. `doi:10.1109/MCSI.2016.054`.

12 R. Ishikawa, E. Chida, and T. Mizuki. Efficient card-based protocols for generating a hidden random permutation without fixed points. In *Proceedings of the 14th International Conference on Unconventional Computation and Natural Computation (UCNC)*, pages 215–226, 2015. `doi:10.1007/978-3-319-21819-9_16`.

13 K. Kotsuma and Y. Takenaga. NP-completeness and enumeration of Number Link puzzle. *IEICE Technical Report*, 109(465):1–7, 2010.

14 J.F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5(3):31–36, 1975. `doi:10.1145/1061425.1061430`.

15 D. Miyahara, T. Sasaki, T. Mizuki, and H. Sone. Card-based physical zero-knowledge proof for Kakuro. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E102.A(9):1072–1078, 2019. `doi:10.1587/transfun.E102.A.1072`.

16 Nikoli: Numberlink. `https://www.nikoli.co.jp/en/puzzles/numberlink.html`.

17 T. Sasaki, T. Mizuki, and H. Sone. Card-based zero-knowledge proof for Sudoku. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN)*, pages 29:1–29:10, 2018. `doi:10.4230/LIPIcs.FUN.2018.29`.

The Computational Complexity of Evil Hangman

Jérémy Barbay
Department of Computer Science, University of Chile, Santiago, Chile
http://barbay.cl
jeremy@barbay.cl

Bernardo Subercaseaux
Department of Computer Science, University of Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
bsuberca@dcc.uchile.cl

Abstract

The game of Hangman is a classical asymmetric two player game in which one player, the setter, chooses a secret word from a language, that the other player, the guesser, tries to discover through single letter matching queries, answered by all occurrences of this letter if any. In the Evil Hangman variant, the setter can change the secret word during the game, as long as the new choice is consistent with the information already given to the guesser. We show that a greedy strategy for Evil Hangman can perform arbitrarily far from optimal, and most importantly, that playing optimally as an Evil Hangman setter is computationally difficult. The latter result holds even assuming perfect knowledge of the language, for several classes of languages, ranging from Finite to Turing Computable. The proofs are based on reductions to Dominating Set on 3-regular graphs and to the Membership problem, combinatorial problems already known to be computationally hard.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Complexity classes

Keywords and phrases combinatorial game theory, computational complexity, decidability, hangman

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.23

Acknowledgements We want to thank Robinson Castro, who first introduced us to this problem. We also thank Nicolás Sanhueza-Matamala, Alex Meiburg and the anonymous reviewers for their insightful comments and discussion.

1 Introduction

The HANGMAN's game is a classical asymmetric two player game, where one player, denoted as *the setter* keeps a secret word w, that the other player, denoted as *the guesser*, needs to guess. The game starts with both players agreeing on a maximum number of guesses d, and the setter communicating to the guesser an integer k, the length of the word w. Then, in every turn the guesser makes a query with a single letter s, and the setter reveals every appearance of s in w, if any. A query s is said to have failed if there are no occurrences of s in w. The game ends either when w is fully revealed, in which case we say the guesser won, or when the guesser has made more than d failed queries, in which case we say the setter won the game.

The EVIL HANGMAN variant is a little twist to the game that has been widely used as a programming assignment [11]. In this variant, the setter can change the secret word as often as she wants during the game, as long as at the end she's able to reveal a word that is consistent with all the answers given thus far to the guesser's queries. In the programming assignment [11], students are given the task of implementing what we call the `GreedyCheater`, an evil setter that decides to answer each query with the heuristic of keeping the dictionary of consistent words to reveal at the end as big as possible.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 23; pp. 23:1–23:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A natural question is whether the algorithm `GreedyCheater` is the best one can do when trying to maximize the number of guesses a guesser requires to discover the secret word, and if it is not, then **what would be the best strategy to maximize the number of questions required to guess a word in a** Evil Hangman **game**. Such questions can be asked in various contexts, such as when the language from which the words are chosen and guessed is a natural one (i.e. a finite set of words of fixed length k), or as when the language is defined more formally (e.g. through a Turing machine, the language being projected to a finite set of words of fixed length k).

We formalize such various contexts of the Hangman and Evil Hangman game, and the related computational problems, in Section 2. As a preliminary result, we show that the number of guesses generated by the `GreedyCheater` strategy is not only sub-optimal on some trivial examples, but that it can be arbitrarily worse than the optimal strategy on an infinite family of scenarios (in Section 3). Our main result is about finding an optimal strategy for the game of Evil Hangman: while it is clear that there is an exponential time *minimax* strategy for playing optimally as an evil setter (see Equation 2), we prove (in Section 4) that such a running time is essentially optimal in the worst case, as the problem of deciding for a given set L of words and a maximal number d of guesses whether there is a strategy forcing more than d failed guesses, is coNP-hard (and hence, deciding if there is a winnning stategy for the guesser is NP-hard). Pushing further such results, we prove (in Section 5) that playing optimally is PSPACE-complete when the game is played over the language defined by a Context Sensitive Grammar, and undecidable when played over the language defined by a Turing Machine. We conclude in Section 6 with a discussion of other minor results and potential open directions for further research.

Throughout this paper, we consider the problem in the context of a perfect guesser, with perfect knowledge of the language.

2 Preliminaries

Before stating our results, we describe more formally the original Hangman game (Section 2.1), the Evil Hangman variant (Section 2.2), and a formalization of how to measure the quality of strategies for the Evil Hangman variant (Section 2.3).

2.1 Hangman

The game starts with a word length, k, and a parameter d stating the number of failed guesses allowed, being agreed upon between the players. A game of Hangman is played over an alphabet Σ initially set to $[1..\sigma]$ and a (potentially infinite) language L projected to words of length k on the remaining alphabet by intersecting it with Σ^k. The alphabet will be progressively reduced during the game to capture the fact that symbols which have already been guessed should not be part of the game anymore. We will use an extra symbol, not present in Σ that denotes a letter not yet revealed. Players often use an underscore (_) for this, but we will use the symbol $\perp$ instead, for better readability.

As the game goes by turns, we can define the state of the game in terms of which letters have been discarded, and what action is taken, on the i-th turn (turns are 1-indexed). The game starts with the setter revealing $M_0 = \perp^k$, which represents what the guesser knows about the word at that point. Then, on the i-th turn, the guesser makes a query with the symbol s_i, and the setter replies with the mask M_i, which is equal to M_{i-1} except possibly for some occurrences of the $\perp$ symbol, that have been changed to s_i. Figure 1a presents an example of a traditional game of Hangman.

1. $w = fun$
2. $M_0 = \bot\bot\bot$
3. $\boldsymbol{s_1 = e}$
4. $M_1 = \bot\bot\bot$
5. $s_2 = n$
6. $M_2 = \bot\bot n$
7. $\boldsymbol{s_3 = a}$
8. $M_3 = \bot\bot n$
9. $s_4 = u$
10. $M_4 = \bot un$
11. $s_5 = f$
12. $\boldsymbol{M_5 = fun}$

(a) Traditional Hangman (guesser wins). Note that s_1 and s_3 are failed guesses.

1. $w_0 = run$
2. $M_0 = \bot\bot\bot$
3. $s_1 = u$
4. $w_1 = run \implies M_1 = \bot u\bot$
5. $s_2 = n$
6. $w_2 = run \implies M_2 = \bot un$
7. $\boldsymbol{s_3 = r}$
8. $w_3 = pun \implies M_3 = \bot un$
9. $\boldsymbol{s_4 = p}$
10. $w_4 = sun \implies M_4 = \bot un$
11. $\boldsymbol{s_5 = s}$
12. $w_5 = fun \implies M_5 = \bot un$

(b) Evil Hangman (setter wins). Note that s_3, s_4 and s_5 are failed guesses.

Figure 1 Example games of Hangman ($k = 3, d = 3$) over the Latin lowercase alphabet and using English as a language.

We define as well the operations $a \oplus b$ to be the result of replacing every $\bot$ character in a by its corresponding character in b, and $(a \cap b)_i$ to be a_i if $a_i = b_i$, and $\bot$ otherwise. Now we state that given a secret word w, we can compute M_i after a guess s_i as $M_{i+1} = M_i \oplus (w \cap s_i^k)$.

It is also helpful to define B_i as the indices in M_i that have the symbol $\bot$, as this is the set that the setter can choose a subset from when answering a query. We use as well the notation s^B, with s being a symbol and B a subset of $[1..k]$, where k is implicit, to describe the word w of size k such that $w_i = s$ if $i \in B$ and $w_i = \bot$ otherwise. Finally, for a language L and a mask M, we abbreviate the set $\{w \in L \mid M \preceq w\}$ as $\mathcal{F}(L, M)$.

2.2 Evil Hangman

In the evil version of the game, the setter can choose to change the secret word, even every turn, as long as the new choice is consistent with the answers given so far. We say that w_i is the secret word before the setter reveals M_i. Figure 1b presents an example of a an Evil Hangman game.

In order to define what the required *consistency* exactly means, we define a relation $\preceq$, such that for two words a and b of length k

$$a \preceq b \iff (a_i = \bot) \vee (a_i = b_i), \quad \text{for all } i \in \{1, \ldots, k\}$$

Intuitively, we say that b is consistent with a, as the only differences they can have are positions where a had not been revealed yet. We can now state our consistency restrictions as follows:

1. $w_i \in L$
2. $M_i \preceq w_i$
3. $w_{i_j} \notin \{s_1, ..., s_{i-1}\}, \forall j \in B_i$

The first rule requires the partial secret words to be part of the language, the second one requires that partial secret words do not differ with the exposed mask, and the third one requires that the symbols not yet revealed to the guesser do not match any of the previous guesses. The latter is simply captured by the (dynamic) alphabet Σ, which contains only the remaining possible guesses.

2.3 Evaluation of Evil Hangman strategies

An evil setter strategy, over a language L, is a function A_L that takes a mask M_i, a guess s_i and returns $A_L(M_i, s_i) = M_{i+1}$ a new mask, such that $M_i \preceq M_{i+1}$, and there exists $w \in L$ such that $M_{i+1} \preceq w$. We define the function W to measure how good a particular situation is for the setter. A situation is a tuple (M, Σ) where M is a word mask and Σ is the remaining alphabet (letters that have not been guessed so far). We define $W(M, \Sigma, A_L)$ for an adversarial setter A_L as the minimum number of failed guesses that any player would have to do in order to reveal a full word, starting from M over the alphabet Σ. We can formalize the function W inductively as follows, where we use the Iverson bracket notation [13], namely, for any predicate P, the expression $[\![P]\!]$ equals 1 if P is true and 0 otherwise:

$$W(M, \Sigma, A_L) = \begin{cases} 0 & \text{if } M \in L \\ \min\limits_{s \in \Sigma} \left\{ W \begin{pmatrix} A_L(M, s), \\ \Sigma \setminus \{s\}, \\ A_{\mathcal{F}(L, A_L(M,s))} \end{pmatrix} + [\![A_L(M, s) = M]\!] \right\} & \text{otherwise} \end{cases} \tag{1}$$

We define OPT as an adversary such that $W(M, \Sigma, OPT)$ is maximum for every M and Σ. Noting B_M the set of indices of a word M containing $\perp$ allows us to state OPT explicitly as a recursive formula:

$$OPT_L(M, s, \Sigma) = \max_{\substack{B \subseteq B_M \\ \mathcal{F}(L, M \oplus s^B) \neq \varnothing}} \left\{ [\![B = \emptyset]\!] + \min_{s' \in \Sigma} \left\{ OPT_{\mathcal{F}(L, M \oplus s^B)} \left(M \oplus s^B, s', \Sigma \setminus \{s\} \right) \right\} \right\} \tag{2}$$

We will refer to this optimal adversary in the next section, when showing that the greedy algorithm's competitive ratio is not bounded by a constant, and thus that such an algorithm can perform arbitrarily bad on an infinite family of instances.

3 The greedy cheater

Algorithm 1 presents a pseudo code for the algorithm `GreedyCheater`, a pretty intuitive and efficient algorithm for the setter, that is often given as a programming assignment [11] in colleges across the US. The idea is to answer every query in such a way that the number of words remaining in the dictionary that are consistent with the answer is maximized.

Not only is the `GreedyCheater` algorithm not optimal, it can be arbitrarily bad, which we formalize in the following theorem, and illustrate with an example in Figure 2.

▶ **Theorem 1.** *`GreedyCheater` is not c-competitive in terms of W. That is, there are no constants $c > 0$ and b such that $c \cdot W(M, \Sigma, \texttt{GreedyCheater}) + b \geq W(M, \Sigma, OPT_L)$ for every possible language L and situation (M, Σ).*

Proof. We describe how to build an adversarial dictionary D of size $n = 2m + 1$, where

- $m + 1$ words start with the symbol α, and have only symbols β and γ in the other positions (both β and γ must be present) and
- the remaining m words are of the form η^k for symbols $\eta \notin \{\alpha, \beta, \gamma\}$.

Note that this requires $k \geq 1 + \lg(m + 1)$ and $\sigma \geq m + 3$, to ensure that we can actually build the first $m + 1$ different words with combinations of β and γ and the last m words with different symbols. This poses no problem, as such k and σ exist for every m, and thus we can build bad instances of arbitrary size.

Algorithm 1 Pseudo code for the `GreedyCheater` algorithm.

```
Input: a mask $M_{i-1}$, a guess $s_i$ and a dictionary $D$
Output: the mask $M_i$
$bestMaskSize \leftarrow 0$
$bestMask \leftarrow NIL$
for $B \in 2^{B_{i-1}}$ do
    $thisMask \leftarrow M_{i-1} \oplus s_i^B$
    $thisMaskSize \leftarrow |\mathcal{F}(D, thisMask)|$
    if $thisMaskSize > bestMaskSize$ then
        $bestMaskSize \leftarrow thisMaskSize$
        $bestMask \leftarrow thisMask$
return $bestMask$
```

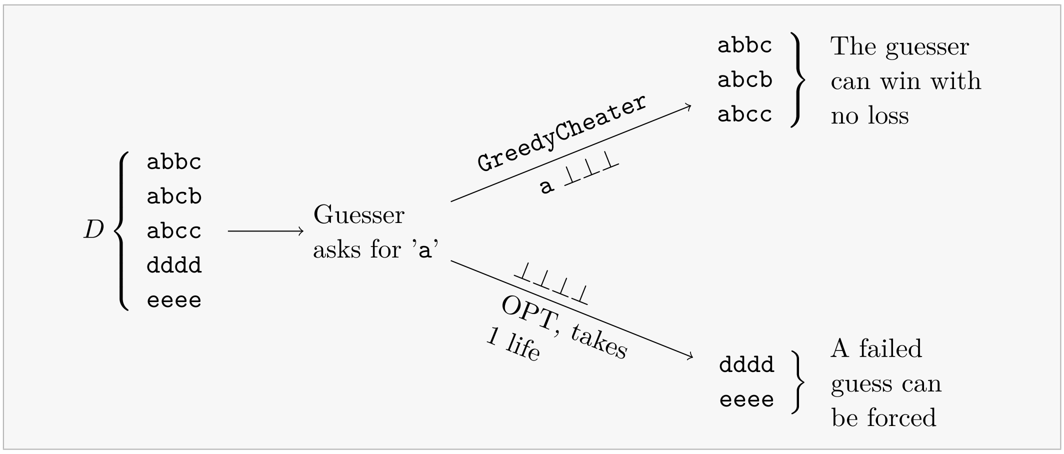


Figure 2 Example of an adversarial dictionary D of size $n = 5$ over the alphabet $\{a, b, c, d, e\}$ against the `GreedyCheater` algorithm, with word length $k = 4$.

Now, upon the guess α, the greedy algorithm will answer $\alpha\bot^{k-1}$ (as $m + 1$ words start with it, as opposed to the m that do not have it), and then after guessing β and γ the guesser will find the word with loss 0.

On the other hand, an optimal Evil Hangman algorithm would reply with $\bot^k$ to the original guess, and then on any guess with a symbol in $\Sigma \setminus \{\alpha, \beta, \gamma\}$ except for the last one, it would reply with the same mask taking a life from the guesser. Such a strategy makes the guesser lose on the symbol α and at least $m - 1$ other symbols, giving a total loss of m. This concludes the proof. ◀

The `GreedyCheater` algorithm can be efficiently implemented. But given that it can perform arbitrarily far from the optimum, a natural question is whether there is an efficient algorithm that achieves optimality, which we explore in the next section.

4 Hardness of Finding an Optimal Evil Adversary

Consider the decision problem of whether the setter can *win* the game against any possible guesser. We restrict our analysis to finite languages in this section, and explore generalized languages (languages that are higher up in the Chomsky hierarchy) in Section 5.

Our main computational problem is, given a finite language L of words of length k on an alphabet of size σ, to decide if an evil Hangman setter has a winning strategy, where winning is defined with respect to the number d of failed guesses that the guesser is allowed:

Evil Hangman

Given a finite language L, where every word has some fixed size k, and an integer d, decide whether it is possible for a cheating setter to play in such a way that no guesser can get the secret word without making at least d failed guesses.

We prove the difficulty of Evil Hangman through a reduction to the problem of Minimum Dominating Vertex Set in 3-regular graphs, by taking such a graph and encoding it as a language. Intuitively, we will build an alphabet by associating a different symbol to each vertex of the graph, and the language will be constructed by associating a word to each vertex. Each symbol will be present in the word of its corresponding vertex and neighbors. Therefore, each time the setter answers a guesser's query negatively (which corresponds to a vertex in the graph), the words associated to its associated vertex and its neighborhood are discarded from the possibles words to reveal at the end, and we can interpret this as discarding that vertex and its neighborhood from the graph. As long as there is a non-discarded vertex in the graph, the setter can claim that the encoding of such a vertex is the secret word. This relates the number of nodes, and their respective neighborhoods, that are enough to cover the entire graph (Minimum Dominating Vertex Set) and the amount of failed guesses a setter can force (Evil Hangman). The relationship between the two problems is however inverted, the existence of a small dominating set will allow the guesser to quickly discard many options and thus will constrain the victory of our protagonist, the setter. This idea leads therefore to a proof of coNP-hardness.

▶ **Theorem 2.** *Evil Hangman is* coNP*-hard, even when restricted to languages with words of length 4.*

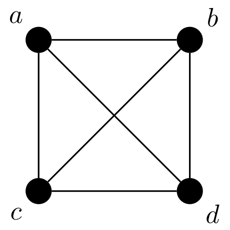
Proof. Given a graph $G = (V, E)$ and a positive integer d, let (G, d) be an instance of the 3-Regular Dominating Set problem (defined formally below)

- Theorem 5 proves that 3-Regular Dominating Set is NP-hard.
- Lemma 9 describes how to compute a language L that properly encodes G in polynomial time. The alphabet size σ of L corresponds to the number of labels in V.
- Lemma 10 proves that if (G, d) is a positive instance of 3-Regular Dominating Set, then (L, d) is a negative instance of Evil Hangman; while
- Lemma 11 proves (by contradiction) that if (L, d) is a positive instance of Evil Hangman, then (G, d) is a negative instance of 3-Regular Dominating Set.

The combination of Lemmas 10 and 11 proves that (G, d) is a positive instance of 3-Regular Dominating Set if and only if (L, d) is a negative instance of Evil Hangman, which permits to deduce the coNP-hardness of Evil Hangman from the NP-hardness of 3-Regular Dominating Set (from Theorem 5) ◀

In order to ensure that the language only contains words of a fixed length, we consider only the restricted class of graphs where the degree of every node is fixed:

▶ **Definition 3** (k-Regular Graph). *A graph $G = (V, E)$ is k-*regular *if and only if every vertex v in V has exactly k neighbors.*



(a) The graph G to encode.

1. $w_a =$ `abcd`
2. $w_b =$ `bacd`
3. $w_c =$ `cabd`
4. $w_d =$ `dabc`

(b) An encoding of G.

1. $w_a =$ `abcd`
2. $w_b =$ `badc`
3. $w_c =$ `cdba`
4. $w_d =$ `dcab`

(c) A proper encoding of G.

Figure 3 Example of encodings for K_4.

The main combinatorial problem required for our results is that of

▶ **Definition 4** (Dominating Vertex Set). *Given a graph $G = (V, E)$, a set of vertices $D \subseteq V$ is* dominating *if and only if every node in V is either a member of D or has a neighbor in D.*

The problem we will reduce from is a restricted version of Minimum Dominating Vertex Set.

3-Regular Dominating Set

Given a 3-regular graph G and an integer d, decide whether the size $\gamma(G)$ of the minimum dominating set is at most d.

The problem of Minimum Dominating Vertex Set has been intensively studied since the 1970s, and its NP-hardness is known for several classes of graphs (Planar, Perfect, Bipartite, Chordal, Split, etc) [1]. Our reduction is based on a result by Kikuno et al. [8], that shows NP-completeness for 3-regular planar graphs. This stronger result implies of course hardness for the broader class of 3-regular graphs, which is essential to our reduction.

▶ **Theorem 5** (Kikuno et al. [8]). *3-Regular Dominating Set is* NP*-hard*

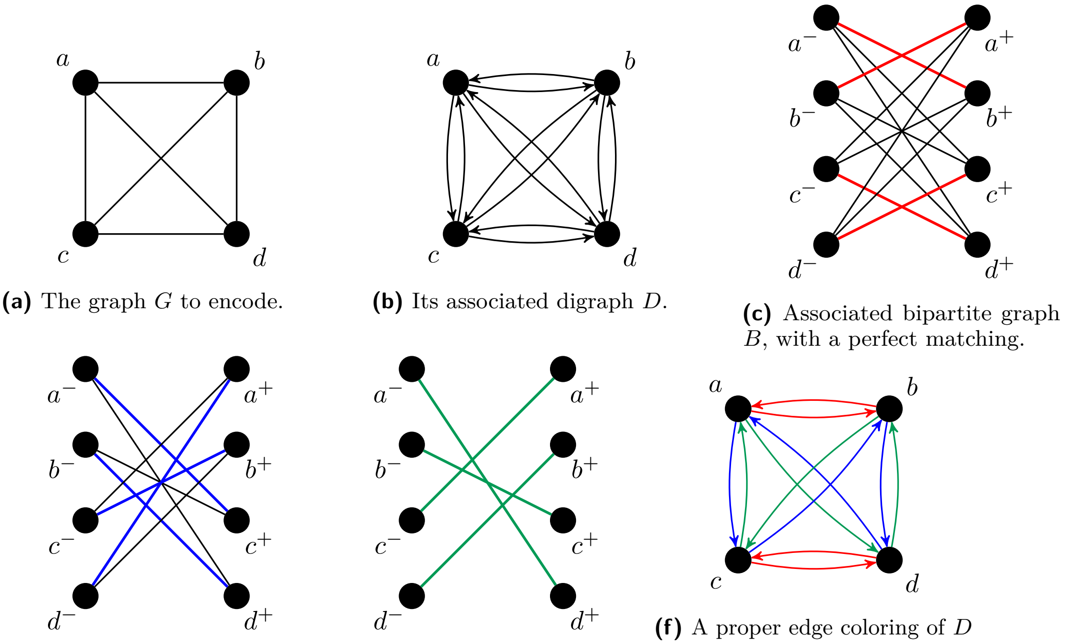
In order to reduce from this problem, we start by showing a proper way to encode a 3-regular graph as a language.

▶ **Definition 6** (Vertex Encoding). *Let $G = (V, E)$ be a 3-regular graph, where every vertex of V is labeled with a symbol from an alphabet Σ. We say that a word of Σ^4 is a* vertex encoding *of a node $v \in V$ if its first symbol is the label of v followed by the labels of its three neighbors.*

Now, by putting together an encoding of every vertex of a graph, we get a graph encoding as a language.

▶ **Definition 7** (Language Encoding a Graph). *Given a graph $G = (V, E)$ whose vertices are labeled with symbols of an alphabet Σ, we say a language $L \subseteq \Sigma^4$* encodes *G if $L = \{w_1, w_2, \ldots, w_{|V|}\}$ where w_i is a vertex encoding of the i-th node.*

An example of such an encoding is presented in Figure 3b. Because vertex encodings can have the neighbors of the represented vertex in any order, there are $(3!)^{|V|}$ possible language encodings for a given 3-regular graph $G = (V, E)$, and they present different combinatorial



(a) The graph G to encode. **(b)** Its associated digraph D. **(c)** Associated bipartite graph B, with a perfect matching.

(d) A second perfect matching. **(e)** Third perfect matching. **(f)** A proper edge coloring of D based on the matchings.

Figure 4 Illustration for the proof of Lemma 9 on K_4. Note that the encoding resulting of subfigure (f) corresponds to the one presented in Figure 3c.

properties. We describe a deterministic way to encode input graphs that permits to identify any word just by knowing of a single letter in it, so that no two words can have the same symbol on the same position. This property greatly simplifies the proof of the reduction in Lemma 11.

▶ **Definition 8** (Proper Graph Encoding). *An encoding L of a graph $G = (V, E)$ is said to be* proper *if for every vertex $v \in V$, and every position p in $\{1, 2, 3, 4\}$, there is exactly one word in which the label of vertex v appears in the p^{th} position.*

In Figure 3c we present an example of a proper encoding. We now prove a key lemma in our reduction: the fact that we can compute a proper encoding of any 3-regular graph in time polynomial in the number of vertices of G.

▶ **Lemma 9.** *Every 3-regular graph G admits a proper encoding, and such an encoding can be computed in polynomial time.*

Proof. Let $G = (V, E)$ be a 3-regular graph, we start by considering the digraph $D = (V', E')$ associated to G where $V' = V$ and E' contains the pairs (u, v) and (v, u) if there was an edge between nodes u and v in G. We claim that if there exists a way to color edges in D with {`red, blue, green`} such that every vertex has (i) incoming edges of each different color, and (ii) outgoing edges of each different color, then we can produce a proper encoding based on that. Here's how to do it: if vertex u has a red outgoing edge to v, a green outgoing edge to w and a blue outgoing edge to x, then we can encode it as `uvwx`. Note that the color of an edge $u \to v$ determines in which position is v going to appear in the encoding of u, and therefore condition (i) over v ensures that the label of v appears in every position, while condition (ii) over u ensures that no more than one vertex is assigned position p on the encoding of u.

In order to find such an edge coloring, we create the undirected bipartite graph $B = (V'', E'')$, where for every vertex $v \in V$, we put two vertices v^+ and v^- in V'', and for every edge (u, v) in E' we put the edges (u^+, v^-) and (u^-, v^+). The partition of B is then, of course, the set of vertices $(\cdot)^+$ and the set of vertices $(\cdot)^-$. Note that B is also a 3-regular graph, as every vertex v with neighbors u, w and x in the original graph, it is associated vertex u^+ is connected with v^-, w^- and x^- in B, and u^- will be connected to u^+, w^+ and x^+.

As a direct consequence of Hall's Marriage Theorem [5], every regular bipartite graph has a perfect matching. Such a perfect matching can be computed in polynomial time using for example the Hopcroft-Karp algorithm [6]. Let M be the set of edges of a perfect matching computed that way. We can color every edge in M with red. Now, if we remove from E'' all the edges of M, the bipartite graph is 2-regular, as each node has lost exactly one neighbor. By using Hall's theorem again, we can get a new perfect matching M', whose edges we color with blue. If we now remove all the edges of M'', we get a 1-regular graph, which is itself a perfect matching, and whose edges we color with green. This is enough to get the required coloring in the graph D, just by coloring every edge (u, v) with the same color of the edge (u^-, v^+). ◀

The final step of the proof of Theorem 2 consists in proving that (G, d) is a positive instance of 3-Regular Dominating Set if and only if (L, d) is a negative instance of Evil Hangman. Lemma 10 proves the forward direction of the statement:

▶ **Lemma 10.** *Let G be a 3-regular graph, and L be the language built from G as described in Lemma 9, and let d be an arbitrary integer. If (G, d) is a positive instance of 3-Regular Dominating Set, then (L, d) is a negative instance of Evil Hangman.*

Proof. Let's assume (G, d) is a positive instance of the 3-Regular Dominating Set problem. This means that there is a dominating set for G of size at most d. Let $\Gamma == \{v_1, v_2, \ldots, v_{\gamma(G)}\}$ be such a dominating set. Then, if the guesser makes the sequence of queries $\ell(v_1), \ell(v_2), \ldots, \ell(v_{\gamma(G)})$, where $\ell(v)$ corresponds to the label of vertex v, the setter is forced to answer positively at least one of those queries, as otherwise there would be no possible word for her to reveal at the end. Thus far, the guesser has made at most $d - 1$ failed queries. As the encoding of the graph G into the language L is proper, a single guess answered positively is enough to uniquely determine the secret word, and therefore the guesser can win the game without making any more failed guesses, implying that the instance (L, d) is negative for Evil Hangman. ◀

Lemma 11 proves (by contradiction) the reverse direction:

▶ **Lemma 11.** *Let G be a 3-regular graph, and L be the language built from G as described in Lemma 9, and let d be an arbitrary integer. If (L, d) is a negative instance of Evil Hangman, then (G, d) is a positive instance of 3-Regular Dominating Set.*

Proof. We will show the contrapositive statement, namely, that if (G, d) is a negative instance of 3-Regular Dominating Set then (L, d) is a positive instance of Evil Hangman. Let's assume that (G, d) is a negative instance, and thus, d vertices are not enough to dominate the graph. This would mean that for any set D of d vertices, there is at least one vertex v_D which is not dominated by D. Consider then that the guesser makes a sequence of d queries, whose associated vertices form the set D'. Then, rejecting all those d queries and revealing $w(v_{D'})$ as the secret word is a guaranteed strategy for the setter, meaning that (L, d) is a positive instance of Evil Hangman. ◀

This concludes the proof of the coNP-hardness of Evil Hangman when L is a finite language. We explore its computational complexity for more general types of languages in the next section.

5 Generalized Languages

A natural generalization of the Evil Hangman problem is to consider its complexity when played over broader classes of languages, such as Regular, Context Free or Turing computable languages over an alphabet Σ, projected to words of length k by intersection with Σ^k. The lower bound of Theorem 2 (where the language is finite) can be extended to prove hardness for the cases where the language is defined by a Regular Expression (or, respectively, a Context Free Grammar or a Turing Machine) by observing that a dictionary of n words of length k can be encoded in a Regular Expression (or respectively, a Context Free Grammar or a Turing Machine) of size $(nk)^{O(1)}$, and thus we can construct hard instances for such problems by using the same construction used to prove Theorem 2.

In this section we give a result for classes of languages whose associated machines are strong enough to simulate other machines within their class. Namely, that when the game is played over languages of such classes, the Evil Hangman decision problem is at least as hard as deciding membership of a word in the language. This implies undecidability for Turing computable languages (given as Turing Machines, abbreviated as TM) and PSPACE-completeness for Context Sensitive Languages [10] (given as their equivalent Linear Bounded Automata [9], abbreviated as LBA). The proofs of hardness are thus based on reductions to the membership problem, stated below.

Membership

Given an encoding of a machine (or language) C belonging to a class $\mathcal{C}$, and a word w, decide whether $w \in L(C)$

We consider first a restricted class of languages among Turing Computable languages:

▶ **Definition 12** (Universal Simulation Languages). *Let $\mathcal{C}$ be a class of machines, we say that $\mathcal{C}$ allows universal simulation if, given a machine C, it is possible to construct in polynomial time a machine C' that accepts exactly the language $\{\alpha, \beta\}$ if $(C, w) \in$ Membership$(\mathcal{C})$ and $\{\alpha\}$ otherwise.*

We can now state the key lemma used to prove undecidability of Evil Hangman(TM) and PSPACE-completeness for Evil Hangman(LBA).

▶ **Lemma 13.** *Let $\mathcal{C}$ be a class of machines (languages) allowing universal simulation. Then, there is a polynomial time reduction problem from Membership$(\mathcal{C})$ to Evil Hangman$(\mathcal{C})$, that is, Evil Hangman but over a language L defined by an element of $\mathcal{C}$.*

Proof. Consider and an arbitrary element $C \in \mathcal{C}$. Because of the universal simulation property, we can construct a machine C' with the behavior specified in Definition 12. Now consider $k = 1$ and the instance $(L(C'), d = 1)$. We can see that is a positive instance of Evil Hangman if and only if C accepts on β, as if the C accepts β the dictionary has size 2 and can force a failed guess, but if C rejects β, the dictionary will have size 1 and thus it is not possible to force a failed guess. ◀

We have now the machinery required to easily prove the following two theorems, that define the computability and complexity of Evil Hangman over Context Sensitive languages and Turing Computable languages. A reduction from Membership(PSPACE) yields the PSPACE completeness:

▶ **Theorem 14.** *Evil Hangman is* PSPACE*-complete when the language L is the language defined by an arbitrary Linear Bounded Automaton M.*

Proof. Membership(LBA) is PSPACE-complete [7]. This implies membership in PSPACE by a naive simulation of Equation (2). To prove hardness, we can use a result by Feldman et al. [3] that states that for every n, there is a universal LBA M_n for the class of LBAs using at most n tape symbols, and thus LBAs hold the property of universal simulation (Definition 12). As Membership(LBA) is in particular PSPACE-hard, the reduction implies as well hardness for our problem. ◀

A similar reduction from Membership, but this time from TM, yields the undecidability result:

▶ **Theorem 15.** *Evil Hangman is undecidable when the language L is the language defined by an arbitrary Turing machine M.*

Proof. We reduce from Membership(TM), directly from Lemma 13, as Turing Machines trivially hold the property of universal simulation from Definition 12. The fact that membership is undecidable for Turing Machines (Rice's Theorem) concludes the proof. ◀

This concludes our results about the computational complexity of optimal strategies for the Evil Hangman problem. In the next section, we summarize our results and outline some remaining open questions.

6 Discussion

On one hand, the greedy strategy for Evil Hangman (the one which is commonly given as a programming assignment) can perform arbitrarily bad on certain languages (Theorem 1); on the other hand finding an optimal strategy for a given language is coNP-hard (Theorem 2), and thus we cannot expect a polynomial time algorithm for it unless P = NP. Note that the coNP-hardness from the setter's perspective implies NP-hardness from the guesser's perspective: Theorem 2 is equivalent to the NP-hardness of deciding whether a guesser can always win the game without making d failed queries. Even worse, the optimality of an answer by the evil setter is PSPACE-complete for languages described by Context Sensitive Grammars (Theorem 14) and undecidable for Turing Computable languages (Theorem 15).

Although hard in arbitrary languages, the game of Hangman is traditionally played on natural languages, where alphabets are pretty small, and words are pretty short. Hence it is worth noticing that Equation 2 (given on page 4) yields a Fixed Parameter Tractable (FPT) algorithm [2, 4] when parameterized over $\ell = \sigma + k$ (size of the alphabet + word length). In particular, it can be implemented in time within $n \cdot 2^{O(\ell)}$, where $n = |L|$. First, note that the recursive formula goes over all the possible σ^k masks, all possible symbols, and all the 2^σ possible subsets of the alphabet. This last term can be immediately optimized by considering only the masks that are present in the dictionary, which are no more than $n2^k$. Note that by considering only those masks, the remaining language (in the subscript of OPT in Equation 2) is kept implicit. Therefore, the total number of cells is bound by $\sigma n 2^\sigma 2^k \in n2^{O(\ell)}$. At every cell we have to choose between at most 2^k sub-masks of M and σ

symbols, and compute f which is done in time within $O(k)$. Thus, the total computational work per cell is within $2^{O(\ell)}$. Multiplying this by the amount of cells gives us the desired result. It is an open problem whether EVIL HANGMAN becomes FPT when parameterized by d, the number of failed guesses allowed. The reduction presented in Lemma 11 constitutes an FPT reduction from MINIMUM DOMINATING VERTEX SET on 3-regular graphs. It is well known that MINIMUM DOMINATING VERTEX SET on general graphs is complete for the class $W[2]$ (Downey et al. [2, 4]). However, k-regular graphs are $K_{k+1,k+1}$ free, and thus the result of Telle et al. [12] implies MINIMUM DOMINATING VERTEX SET to be FPT when parameterized by the size of the set. This of course does not imply that EVIL HANGMAN is FPT under such a parameterization: only that we cannot derive fixed parameter intractability from the reduction to dominating set in 3-regular graphs presented in Lemmas 10 and 11.

References

1 Derek G. Corneil and Lorna K. Stewart. Dominating sets in perfect graphs. In *Topics on Domination*, pages 145–164. Elsevier, 1991.

2 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, April 1995.

3 Eliot D. Feldman and James C. Owings. A class of universal linear bounded automata. *Inf. Sci.*, 6:187–190, 1973.

4 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

5 P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, January 1935.

6 John E. Hopcroft and Richard M. Karp. An $n^{5/2} $ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.

7 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972.

8 Tohru Kikuno, Noriyoshi Yoshida, and Yoshiaki Kakuda. The np-completeness of the dominating set problem in cubic planer graphs. *IEICE TRANSACTIONS (1976-1990)*, 63(6):443–444, 1980.

9 S.-Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, June 1964.

10 Peter Linz. *An Introduction to Formal Languages and Automata, Fifth Edition.* Jones and Bartlett Publishers, Inc., USA, 5th edition, 2011.

11 Nick Parlante, Julie Zelenski, Keith Schwarz, Dave Feinberg, Michelle Craig, Stuart Hansen, Michael Scott, and David J. Malan. Nifty assignments. In *Proceedings of the 42nd ACM technical symposium on Computer Science Education - SIGCSE '11*. ACM Press, 2011.

12 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *Algorithms – ESA 2012*, pages 802–812. Springer Berlin Heidelberg, 2012.

13 Wikipedia.org. Iverson bracket notation. `https://en.wikipedia.org/wiki/Iverson_bracket`. Last accessed on 2020-02-06.

Singletons for Simpletons: Revisiting Windowed Backoff with Chernoff Bounds

Qian M. Zhou
Mississippi State University, Department of Mathematics and Statistics, MS, USA
qz70@msstate.edu

Aiden Calvert
Mississippi School for Mathematics and Science, Columbus, MS, USA
calverta20@themsms.org

Maxwell Young
Mississippi State University, Department of Computer Science and Engineering, MS, USA
myoung@cse.msstate.edu

Abstract

Backoff algorithms are used in many distributed systems where multiple devices contend for a shared resource. For the classic balls-into-bins problem, the number of singletons – those bins with a single ball – is important to the analysis of several backoff algorithms; however, existing analyses employ advanced probabilistic tools to obtain concentration bounds. Here, we show that standard Chernoff bounds can be used instead, and the simplicity of this approach is illustrated by re-analyzing some well-known backoff algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Chernoff bounds, backoff, contention resolution, algorithms

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.24

Funding *Maxwell Young*: This research is supported by the National Science Foundation grant CNS 1816076, and by the U.S. National Institute of Justice (NIJ) Grant 2018-75-CX-K002.

1 Introduction

Backoff algorithms address the general problem of how to share a resource among multiple devices [38]. A ubiquitous application is IEEE 802.11 (WiFi) networks [31, 48, 34], where the resource is a wireless channel, and devices each have packets to send. Any single packet sent uninterrupted over the channel is likely to be received, but if the sending times of two or more packets overlap, communication often fails due to destructive interference at the receiver (i.e., a collision). An important performance metric is the time required for all packets to be sent, which is known as the ***makespan***.

Formal Model. Time is discretized into ***slots***, and each packet can be transmitted within a single slot. Starting from the first slot, a ***batch*** of ***n packets*** is ready to be transmitted on a shared channel. This case, where all packets start at the same time, is sometimes referred to as the ***batched-arrivals*** setting. Each packet can be viewed as originating from a different source device, and going forward we speak only of packets rather than devices.

For any fixed slot, if a single packet sends, then the packet ***succeeds***; however, if two or more packets send, then all corresponding packets ***fail***. A packet that attempts to send in a slot learns whether it succeeded and, if so, the packet takes no further action; otherwise, the packet learns that it failed in that slot, and must try again at a later time.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 24; pp. 24:1–24:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Background on Analyzing Makespan. A natural question is the following: *For a given backoff algorithm under batched-arrivals, what is the makespan as measured in the number of slots?*

This question was first addressed by Bender et al. [5] who analyze several backoff algorithms that execute over disjoint, consecutive sets of slots called ***windows***. In every window, each packet that has not yet succeeded selects a single slot uniformly at random in which to send. If the packet succeeds, then it leaves the system; otherwise, the failed packet waits for the next window to begin and repeats this process.

Bender et al. [5] analyze several algorithms where windows monotonically increase in size. The well-known ***binary exponential backoff*** algorithm – a critical component of many WiFi standards – exemplifies this behavior, where each successive window increases in size by a factor of 2.[1]

There is a close relationship between the execution of such algorithms in a window, and the popular balls-in-bins scenario, where $\boldsymbol{N}$ balls (corresponding to packets) are dropped uniformly at random into $\boldsymbol{B}$ bins (corresponding to slots). In this context, we are interested in the number of bins containing a single ball, which are sometimes referred to as ***singletons*** [52].

Despite their simple specification, windowed backoff algorithms are surprisingly intricate in their analysis. In particular, obtaining concentration bounds on the number of slots (or bins) that contain a single packet (or ball) – which we will *also* refer to as singletons – is complicated by dependencies that rule out a straightforward application of Chernoff bounds (see Section 2.1). This is unfortunate given that Chernoff bounds are often one of the first powerful probabilistic tools that researchers learn, and they are standard material in a randomized algorithms course.

In contrast, the makespan results in Bender et al. [5] are derived via delay sequences [33, 49], which are arguably a less-common topic of instruction. Alternative tools for handling dependencies include Poisson-based approaches by Mizenmacher [40] and Mitzenmacher and Upfal [39], and the Doob martingale [21], but to the best of our knowledge, these have not been applied to the analysis of windowed backoff algorithms.

1.1 Our Goal

Is there a simpler route to arrive at makespan results for windowed backoff algorithms?

Apart from being a fun theoretical question to explore, an affirmative answer might improve accessibility to the area of backoff algorithms for researchers. More narrowly, this might benefit students embarking on research, many of whom cannot fully appreciate the very algorithms that enable, for example, their ~~Instagram posts~~ access to online course notes.[2] Arguably, Chernoff bounds can be taught without much setup. For example, Dhubashi and Panconesi [21] derive Chernoff bounds starting on page 3, while their discussion of concentration results for dependent variables is deferred until Chapter 5.

What if we could deploy standard Chernoff bounds to analyze singletons? Then, the analysis distills to proving the correctness of a "guess" regarding a recursive formula (a well-known procedure for students) describing the number of packets remaining after each window, and that guess would be accurate with small error probability.

[1] In practice, the doubling terminates at some fixed large value set by the standard.

[2] In our experience, the makespan analysis is inaccessible to most students in the advanced computer networking course.

Finally, while it may not be trivial to show that Chernoff bounds are applicable to backoff, showing that another problem – especially one that has such important applications – succumbs to Chernoff bounds is aesthetically satisfying.

1.2 Results

We show that Chernoff bounds can indeed be used as proposed above. Our approach involves an argument that the indicator random variables for counting singletons satisfy the following property from [22]:

▶ **Property 1.** *Given a set of n indicator random variables $\{X_1, \cdots, X_n\}$, for all subsets $\$ \subset \{1, \cdots, n\}$ the following is true:*

$$Pr\left[\bigwedge_{j\in\$} X_j = 1\right] \leq \prod_{j\in\$} Pr\left[X_j = 1\right]. \tag{1}$$

We prove the following:

▶ **Theorem 1.** *Consider N balls dropped uniformly at random into B bins. Let $I_j = 1$ if bin j contains exactly 1 ball, and $I_j = 0$ otherwise, for $j = 1, \cdots, B$. If $B \geq N + \sqrt{N}$ or $B \leq N - \sqrt{N}$, then $\{I_1, \cdots, I_B\}$ satisfy the Property 1.*

Property 1 permits the use of standard Chernoff bounds; this implication is posed as an exercise by Dubhashi and Panconesi [21] (Problem 1.8), and we provide the argument in our appendix.

We then show how to use Chernoff bounds to obtain asymptotic makespan results for some of the algorithms previously analyzed by Bender et al. [5]: Binary Exponential Backoff (BEB), Fixed Backoff (FB), and Log-Log Backoff (LLB). Additionally, we re-analyze the asymptotically-optimal (non-monotonic) Sawtooth Backoff (STB) from [29, 25].

These algorithms are specified in Section 5, but our makespan results are stated below.

▶ **Theorem 2.** *For a batch of n packets, the following holds with probability at least $1-O(1/n)$:*

- *FB has makespan at most $n \lg\lg n + O(n)$.*
- *BEB has makespan at most $512n \lg n + O(n)$.*
- *LLB has makespan $O(n \lg\lg n/ \lg\lg\lg n)$.*
- *STB has makespan $O(n)$.*

We highlight that both of the cases in Theorem 1, $B \leq N + \sqrt{N}$ and $B \geq N - \sqrt{N}$, are useful. Specifically, the analysis for BEB, FB, and STB uses the first case, while LLB uses both.

1.3 Related Work

Several prior results address dependencies and their relevance to Chernoff bounds and load-balancing in various balls-in-bins scenarios. In terms of backoff, the literature is vast. In both cases, we summarize closely-related works.

Dependencies, Chernoff Bounds, & Ball-in-Bins. Backoff is closely-related to balls-and-bins problems [4, 18, 47, 50], where balls and bins correspond to packets and slots, respectively. Balls-in-bins analysis often arises in problems of load balancing (for examples, see [9, 10, 11]).

Dubhashi and Ranjan [22] prove that the occupancy numbers – random variables N_i denoting the number of balls that fall into bin i – are negatively associated. This result is used by Lenzen and Wattenhofer [35] use it to prove negative association for the random variables that correspond to at most $k \geq 0$ balls.

Czumaj and Stemann [19] examine the maximum load in bins under an adaptive process where each ball is placed into a bin with minimum load of those sampled prior to placement. Negative association of the occupancy numbers is important to this analysis.

Finally, Dubhashi and Ranjan [22] also show that Chernoff bounds remain applicable when the corresponding indicator random variables that are negatively associated. The same result is presented in Dubhashi and Panconesi [21].

Backoff Algorithms. Many early results on backoff are given in the context of statistical queuing-theory (see [30, 28, 43, 26, 30, 27]) where a common assumption is that packet-arrival times are Poisson distributed.

In contrast, for the batched-arrivals setting, the makespan of backoff algorithms with monotonically-increasing window sizes has been analyzed in [5], and with packets of different sizes in [6]. A windowed, but non-monotonic backoff algorithm which is asymptotically optimal in the batched-arrival setting is provided in [25, 29, 2].

A related problem is *contention resolution*, which addresses the time until the first packet succeeds [51, 41, 24, 23]. This has close ties to the well-known problem of leader election (for examples, see [13, 12]).

Several results examine the *dynamic* case where packets arrive over time as scheduled in a worst-case fashion [36, 20, 8]; this is in contrast to batched-arrivals where it is implicitly assumed that the current batch of packets succeeds before the next batch arrives. A similar problem is that of *wake-up* [16, 15, 17, 14, 37, 32], which addresses how long it takes for a single transmission to succeed when packets arrive under the dynamic scenario.

Finally, several results address the case where the shared communication channel is unavailable at due to malicious interference [3, 44, 45, 46, 42, 1, 7].

2 Analysis for Property 1

We present our results on Property 1. Since we believe this result may be useful outside of backoff, our presentation in this section is given in terms of the well-known balls-in-bins terminology, where we have $\boldsymbol{N}$ balls that are dropped uniformly at random into $\boldsymbol{B}$ bins.

2.1 Preliminaries

Throughout, we often employ the following inequalities (see Lemma 3.3 in [46]), and we will refer to the left-hand side (LHS) or right-hand side (RHS) when doing so.

▶ **Fact 1.** *For any* $0 < x < 1$, $e^{-x/(1-x)} \leq 1 - x \leq e^{-x}$.

Knowing that indicator random variables (i.r.v.s) satisfy Property 1 is useful since the following Chernoff bounds can then be applied.

▶ **Theorem 3.** *(Dubhashi and Panconesi [21])*[3] *Let* $X = \sum_i X_i$ *where* $X_1, ..., X_m$ *are i.r.v.s that satisfy Property 1 . For* $0 < \epsilon < 1$*, the following holds:*

$$Pr[X > (1+\epsilon)E[X]] \quad \leq \quad \exp\left(-\frac{\epsilon^2}{3}E[X]\right) \tag{2}$$

$$Pr[X < (1-\epsilon)E[X]] \quad \leq \quad \exp\left(-\frac{\epsilon^2}{2}E[X]\right) \tag{3}$$

We are interested in the i.r.v.s I_j, where:

$$I_j = \begin{cases} 1, & \text{if bin } j \text{ contains exactly 1 ball.} \\ 0, & \text{otherwise.} \end{cases}$$

Unfortunately, there are cases where the I_js fail to satisfy Property 1. For example, consider $N = 2$ balls and $B = 2$ bins. Then, $Pr(I_1 = 1) = Pr(I_2 = 1) = 1/2$, so $Pr(I_1 = 1) \cdot Pr(I_2 = 1) = 1/4$, but $Pr(I_1 = 1 \wedge I_2 = 1) = 1/2$.

A naive approach (although, we have not seen it in the literature) is to leverage the result in [35], that the variables used to count the number of bins with at most k balls are negatively associated. We may bound the number of bins that have at most 1 ball, and the number of bins that have (at most) 0 balls, and then take the difference. However, this is a cumbersome approach, and our result is more direct.

Returning briefly to the context of packets and time slots, another approach is to consider a subtly-different algorithm where a packet sends with probability $1/w$ in each slot of a window with w slots, rather than selecting uniformly at random a single slot to send in. However, as Bender et al. [5] point out, when n is within a constant factor of the window size, there is a constant probability that the packet will not send in *any* slot. Consequently, the number of windows required for all packets to succeed increases by a $\log n$-factor, whereas only $O(\log\log n)$ windows are required under the model used here.

2.2 Property 1 and Bounding Singletons

To prove Theorem 1, we establish the following Lemma 4. For $j = 1, \cdots, B-1$, define:

$$\mathcal{P}_j = Pr\left[I_{j+1} = 1 \mid I_1 = 1, \cdots, I_j = 1\right]$$

which is the conditional probability that bin $j+1$ contains exactly 1 ball given each of the bins $\{1, \cdots, j\}$ contains exactly 1 ball. Note that $Pr[I_j = 1]$ is same for any $j = 1, \cdots, B$, and let:

$$\mathcal{P}_0 \triangleq Pr[I_j = 1] = N\left(\frac{1}{B}\right)\left(1 - \frac{1}{B}\right)^{N-1}. \tag{4}$$

▶ **Lemma 4.** *If* $B \geq N + \sqrt{N}$ *or* $B \leq N - \sqrt{N}$*, the conditional probability* $\mathcal{P}_j$ *is a monotonically non-increasing function of* j*, i.e.,* $\mathcal{P}_j \geq \mathcal{P}_{j+1}$*, for* $j = 0, \cdots, B-2$*.*

Proof. First, for $j = 1, \cdots, \min\{B, N\} - 1$, the conditional probability can be expressed as

$$\mathcal{P}_j = (N-j)\left(\frac{1}{B-j}\right)\left(1 - \frac{1}{B-j}\right)^{N-j-1}. \tag{5}$$

[3] This is stated in Problem 1.8 in [21]; we present a proof in Section A of our appendix.

Note that $\mathcal{P}_0$ in (4) is equal to (5) with $j = 0$.

For $B \geq N + \sqrt{N}$, we note that beyond the range $j = 1, ..., , \min\{B, N\} - 1$ (i.e., $N - 1$), it must be that $\mathcal{P}_j = 0$. In other words, $\mathcal{P}_j = 0$ for $j = N, N + 1, \cdots, B - 1$ since all balls have already been placed. Thus, we need to prove $\mathcal{P}_j \geq \mathcal{P}_{j+1}$, for $j = 0, \cdots, N - 2$.

On the other hand, if $B \leq N - \sqrt{N}$, we need to prove $\mathcal{P}_j \geq \mathcal{P}_{j+1}$, for $j = 0, \cdots, B - 2$. Thus, this lemma is equivalent to prove if $B \geq N + \sqrt{N}$ or $B \leq N - \sqrt{N}$, the ratio $\mathcal{P}_j / \mathcal{P}_{j+1} \geq 1$, for $j = 0, \cdots, \min\{B, N\} - 2$.

Using the expression (5), the ratio can be expressed as

$$\begin{aligned}
\frac{\mathcal{P}_j}{\mathcal{P}_{j+1}} &= \frac{(N-j)\left(\frac{1}{B-j}\right)\left(1 - \frac{1}{B-j}\right)^{N-j-1}}{(N-j-1)\left(\frac{1}{B-j-1}\right)\left(1 - \frac{1}{B-j-1}\right)^{N-j-2}} \\
&= \frac{1}{\left(\frac{B-j}{N-j}\right)\left(\frac{N-j-1}{B-j-1}\right)} \cdot \frac{\left(1 - \frac{1}{B-j}\right)^{N-j-1}}{\left(1 - \frac{1}{B-j-1}\right)^{N-j-2}} \\
&= \frac{1}{\left(\frac{B-j}{N-j}\right)\left(\frac{N-j-1}{B-j-1}\right)} \cdot \frac{\left(\frac{B-j-1}{B-j}\right)^{N-j-1}}{\left(\frac{B-j-2}{B-j-1}\right)^{N-j-1}\left(\frac{B-j-1}{B-j-2}\right)} \\
&= \frac{1}{\left(\frac{B-j}{N-j}\right)\left(\frac{N-j-1}{B-j-2}\right)} \cdot \left(\frac{\frac{B-j-1}{B-j}}{\frac{B-j-2}{B-j-1}}\right)^{N-j-1} \\
&= \frac{\left(1 + \frac{1}{(B-j)(B-j-2)}\right)^{N-j-1}}{\frac{(N-j-1)(B-j)}{(N-j)(B-j-2)}}.
\end{aligned}$$

Let $a = N - j$, then $2 \leq a \leq N$; and let $y = B - N$. Thus, the ratio becomes

$$\frac{\mathcal{P}_j}{\mathcal{P}_{j+1}} = \frac{\left[1 + \frac{1}{(a+y)(a+y-2)}\right]^{a-1}}{\frac{(a-1)(a+y)}{a(a+y-2)}}.$$

By the Binomial theorem, we have

$$\left[1 + \frac{1}{(a+y)(a+y-2)}\right]^{a-1} = 1 + \frac{a-1}{(a+y)(a+y-2)} + \sum_{k=2}^{a-1} \binom{a-1}{k} \left[\frac{1}{(a+y)(a+y-2)}\right]^k.$$

Thus, the ratio can be written as:

$$\begin{aligned}
\frac{\mathcal{P}_j}{\mathcal{P}_{j+1}} &= \frac{a(a+y-2)}{(a-1)(a+y)} + \frac{a}{(a+y)^2} + \frac{\sum_{k=2}^{a-1}\binom{a-1}{k}\left[\frac{1}{(a+y)(a+y-2)}\right]^k}{\frac{(a-1)(a+y)}{a(a+y-2)}} \\
&= \frac{a^3 + 2a^2y - a^2 + ay^2 - 2ay - a}{a^3 + 2a^2y - a^2 + ay^2 - 2ay - y^2} + \frac{\sum_{k=2}^{a-1}\binom{a-1}{k}\left[\frac{1}{(a+y)(a+y-2)}\right]^k}{\frac{(a-1)(a+y)}{a(a+y-2)}} \\
&= \frac{a^3 + 2a^2y - a^2 + ay^2 - 2ay - a + (y^2 - y^2)}{a^3 + 2a^2y - a^2 + ay^2 - 2ay - y^2} + \frac{\sum_{k=2}^{a-1}\binom{a-1}{k}\left[\frac{1}{(a+y)(a+y-2)}\right]^k}{\frac{(a-1)(a+y)}{a(a+y-2)}} \\
&= 1 + \frac{y^2 - a}{(a+y)^2(a-1)} + \frac{\sum_{k=2}^{a-1}\binom{a-1}{k}\left[\frac{1}{(a+y)(a+y-2)}\right]^k}{\frac{(a-1)(a+y)}{a(a+y-2)}}. \qquad (6)
\end{aligned}$$

Note that because $0 \leq j \leq \min\{B, N\} - 2$, then $a + y = B - j \geq 2$. Thus, the third term in (6) is always non-negative. If $y = B - N \geq \sqrt{N}$ or $y \leq -\sqrt{N}$, then $y^2 \geq N \geq a$ for any $2 \leq a \leq N$. Consequently, the ratio $\mathcal{P}_j / \mathcal{P}_{j+1} \geq 1$. ◀

We can now give our main argument:

Proof of Theorem 1. Let s denote the size of the subset $\mathbb{S} \subset \{1, \cdots, B\}$, i.e. the number of bins in $\mathbb{S}$. First, note that if $B \geq N + \sqrt{N}$, when $s > N$ (i.e., more bins than balls), the probability on the left hand side (LHS) of (1) is 0, thus, the inequality (1) holds. In addition, shown above $Pr[I_j = 1] = \mathcal{P}_0$ for any $j = 1, \cdots, B$. Thus, the right hand side of (1) becomes $\mathcal{P}_0^s$. Thus, we need to prove for any subset, denoted as $\mathbb{S} = \{j_1, \cdots, j_s\}$ with $1 \leq s \leq \min\{B, N\}$

$$Pr\left[\bigwedge_{k=1}^{s} I_{j_k} = 1\right] \leq \mathcal{P}_0^s.$$

The LHS can be written as:

$$\begin{aligned} = \quad & Pr\left[I_{j_s} = 1 \mid \bigwedge_{k=1}^{s-1} I_{j_k} = 1\right] Pr\left[\bigwedge_{k=1}^{s-1} I_{j_k} = 1\right] \\ & = \mathcal{P}_{s-1} Pr\left[\bigwedge_{k=1}^{s-1} I_{j_k} = 1\right] \end{aligned}$$

$$\begin{aligned} & = \mathcal{P}_{s-1} Pr\left[I_{j_{s-1}} = 1 \mid \bigwedge_{k=1}^{s-2} I_{j_k} = 1\right] Pr\left[\bigwedge_{k=1}^{s-2} I_{j_k} = 1\right] \\ & \qquad = \mathcal{P}_{s-1}\mathcal{P}_{s-2} Pr\left[\bigwedge_{k=1}^{s-2} I_{j_k} = 1\right] \\ & \qquad\qquad \vdots \\ & = \mathcal{P}_{s-1}\mathcal{P}_{s-2} \cdots \mathcal{P}_0 \end{aligned}$$

Lemma 4 shows that if $B \geq N + \sqrt{N}$ or $B \leq N - \sqrt{N}$, $\mathcal{P}_j$ is a non-increasing function of $j = 0, \cdots, B - 1$. Consequently, $\mathcal{P}_0 \geq \mathcal{P}_j$, for $j = 1, \cdots, B - 1$. Thus:

$$Pr\left[\bigwedge_{k=1}^{s} I_{j_k} = 1\right] \leq \mathcal{P}_0^s,$$

and so the bound in Equation (1) holds. ◀

The standard Cheroff bounds of Theorem 3 now apply, and we use them obtain bounds on the number of singletons. For ease of presentation, we occasionally use $\exp(x)$ to denote e^x.

▶ **Lemma 5.** *For N balls that are dropped into B bins where $B \geq N + \sqrt{N}$ or $B \leq N - \sqrt{N}$, the following is true for any $0 < \epsilon < 1$.*

- *The number of singletons is at least $\frac{(1-\epsilon)N}{e^{N/(B-1)}}$ with probability at least $1 - e^{\frac{-\epsilon^2 N}{2\exp(N/(B-1))}}$.*
- *The number of singletons is at most $\frac{(1+\epsilon)N}{e^{(N-1)/B}}$ with probability at least $1 - e^{\frac{-\epsilon^2 N}{3\exp(N/(B-1))}}$.*

Proof. We begin by calculating the expected number of singletons. Let I_i be an indicator random variable such that $I_i = 1$ if bin i contains a single ball; otherwise, $I_i = 0$. Note that:

$$\begin{aligned} Pr(I_i = 1) &= \binom{N}{1}\left(\frac{1}{B}\right)\left(1-\frac{1}{B}\right)^{N-1} \\ &\geq \binom{N}{1}\left(\frac{1}{B}\right)\left(1-\frac{1}{B}\right)^{N} \\ &\geq \frac{N}{Be^{(N/(B-1))}} \end{aligned} \quad (7)$$

where the last line follows from the LHS of Fact 1. Let $I = \sum_{i=1}^{B} I_i$ be the number of singletons. We have:

$$\begin{aligned} E[I] &= \sum_{i=1}^{B} E[I_i] \quad \text{by linearity of expectation} \\ &\geq \frac{N}{e^{(N/(B-1))}} \quad \text{by Equation (7)} \end{aligned}$$

Next, we derive a concentration result around this expected value. Since $B \geq N + \sqrt{N}$ or $B \leq N - \sqrt{N}$, Theorem 1 guarantees that the I_is are negatively associated, and we may apply the Chernoff bound in Equation 3 to obtain:

$$Pr\left(I < (1-\epsilon)\frac{N}{e^{(N/(B-1))}}\right) \leq \exp\left(-\frac{\epsilon^2 N}{2e^{(N/(B-1))}}\right)$$

which completes the lower-bound argument. The upper bound is nearly identical. ◀

3 Bounding Remaining Packets

In this section, we derive tools for bounding the number of packets that remain as we progress from one window to the next.

All of our results hold for sufficiently large $n > 0$. Let $\boldsymbol{w_i}$ denote the number of slots in window $i \geq 0$. Let $\boldsymbol{m_i}$ be the number of packets at the start of window $i \geq 0$.

We index windows starting from 0, but this does not necessarily correspond to the initial window executed by a backoff algorithm. Rather, in our analysis, window 0 corresponds to the first window where packets start to succeed in large numbers; this is different for different backoff algorithms.

For example, BEB's initial window consists of a single slot, and does not play an important role in the makespan analysis. Instead, we apply Chernoff bounds once the window size is at least $n + \sqrt{n}$, and this corresponds to window 0. In contrast, for FB, the first window (indeed, *each* window) has size $\Theta(n)$, and window 0 is indeed this first window for our analysis. This indexing is useful for our inductive arguments presented in Section 4.

3.1 Analysis

Our method for upper-bounding the makespan operates in three stages. First, we apply an inductive argument – employing Case 1 in Corollary 6 below – to cut down the number of packets from n to less than $n^{0.7}$. Second, Case 2 of Corollary 6 is used whittle the remaining packets down to $O(n^{0.4})$. Third, we hit the remaining packets with a constant number of calls to Lemma 7; this is the essence of Lemma 8.

Intuition for Our Approach. There are a couple things worth noting. To begin, why not carry the inductive argument further to reduce the number of packets to $O(n^{0.4})$ directly (i.e., skip the second step above)? Informally, our later inductive arguments show that m_{i+1} is roughly at most $n/2^{2^i}$, and so $i \approx \lg\lg(n)$ windows should be sufficient. However, $\lg\lg(n)$ is not necessarily an integer and we may need to take its floor. Given the double exponential, taking the floor (subtracting 1) results in $m_{i+1} \geq \sqrt{n}$. Therefore, the equivalent of our second step will still be required. Our choice of $n^{0.7}$ is not the tightest, but it is chosen for simpicity.

The second threshold of $O(n^{0.4})$ is also not completely arbitrary. In the (common) case where $w_0 \geq n + \sqrt{n}$, note that we require $O(n^{1/2-\delta})$ packets remaining, for some constant $\delta > 0$, in order to get a useful bound from Lemma 7. It is possible that after the inductive argument, that this is already satisfied; however, if not, then Case 2 of Corollary 6 enforces this. Again, $O(n^{0.4})$ is chosen for ease of presentation; there is some slack.

▶ **Corollary 6.** *For $w_i \geq n + \sqrt{n}$, the following is true with probability at least $1 - 1/n^2$:*

- *Case 1. If $m_i \geq n^{7/10}$, then $m_{i+1} < \frac{(5/4)m_i^2}{n}$.*
- *Case 2. If $n^{0.4} \leq m_i < n^{7/10}$, then $m_{i+1} = O(n^{2/5})$.*

Proof. For Case 1, we apply the first result of Lemma 5 with $\epsilon = \frac{\sqrt{4e\ln n}}{n^{1/3}}$, which implies with probability at least $1 - \exp(-\frac{4e\ln n}{n^{2/3}}\frac{n^{0.7}}{2}) \geq 1 - \exp(-2\ln n) \geq 1 - 1/n^2$:

$$\begin{aligned}
m_{i+1} &\leq m_i - \frac{(1-\epsilon)m_i}{e^{m_i/(w_i-1)}} \\
&\leq m_i\left(1 - \frac{1}{e^{m_i/(w_i-1)}} + \epsilon\right) \\
&\leq m_i\left(\frac{m_i}{w_i - 1} + \epsilon\right) \quad \text{by RHS of Fact 1} \\
&\leq \frac{m_i^2}{n} + m_i\epsilon \quad \text{since } w_i \geq n + \sqrt{n} \qquad (8)\\
&\leq \frac{m_i^2}{n} + \left(\frac{m_i}{n^{1/3}}\right)\sqrt{4e\ln n} \\
&< \frac{(5/4)m_i^2}{n} \quad \text{since } m_i \geq n^{7/10}
\end{aligned}$$

where 5/4 is chosen for ease of presentation.

For Case 2, we again apply the first result of Lemma 5, but with $\epsilon = \sqrt{\frac{4e\ln n}{m}}$. Then, with probability at least $1 - 1/n^2$, the first and second terms in Equation 8 are at most $n^{0.4}$ and $O(n^{0.35}\sqrt{\ln n})$, respectively, for the any $n^{0.4} \leq m_i \leq n^{7/10}$. ◀

The following lemma is useful for achieving a with-high-probability guarantee when the number of balls is small relative to the number of bins.

▶ **Lemma 7.** *Assume $w_i > 2m_i$. With probability at least $1 - \frac{m_i^2}{w_i}$, all packets succeed in window i.*

Proof. Consider placements of packets in the window that yield at most one packet per slot. Note that once a packet is placed in a slot, there is one less slot available for each remaining packet yet to be placed. Therefore, there are $w_i(w_i - 1)\cdots(w_i - m_i + 1)$ such placements.

Since there are $w_i^{m_i}$ ways to place m_i packets in w_i slots, it follows that the probability that each of the m_i packets chooses a different slot is:

$$\frac{w_i(w_i - 1)\cdots(w_i - m_i + 1)}{w_i^{m_i}}.$$

We can lower bound this probability:

$$\begin{aligned}
&= \frac{w_i^{m_i}(1-1/w_i)\cdots(1-(m_i-1)/w_i)}{w_i^{m_i}} \\
&\geq e^{-\sum_{j=1}^{m_i-1}\frac{j}{w_i-j}} \quad \text{by LHS of Fact 1} \\
&\geq e^{-\sum_{j=1}^{m_i-1}\frac{2j}{w_i}} \quad \text{since } w_i > 2m_i > 2j \text{ which} \\
&\qquad\qquad\qquad \text{leads to } \frac{j}{w_i-j} < \frac{2j}{w_j} \\
&= e^{-(1/w_i)(m_i-1)m_i} \quad \text{by sum of natural numbers} \\
&\geq 1 - \frac{m_i^2}{w_i} + \frac{m_i}{w_i} \quad \text{by RHS of Fact 1} \\
&> 1 - \frac{m_i^2}{w_i}
\end{aligned}$$

as claimed. ◀

▶ **Lemma 8.** *Assume a batch of $m_i < n^{7/10}$ packets that execute over a window of size w_i, where $w_i \geq n + \sqrt{n}$ for all i. Then, with probability at least $1 - O(1/n)$, any monotonic backoff algorithm requires at most 6 additional windows for all remaining packets to succeed.*

Proof. If $m_i \geq n^{0.4}$, then Case 2 of Corollary 6 implies $m_{i+1} = O(n^{0.4})$; else, we do not need to invoke this case. By Lemma 7, the probability that any packets remain by the end of window $i+1$ is $O(n^{0.8}/n) = O(1/n^{0.2})$; refer to this as the probability of failure. Subsequent windows increase in size monotonically, while the number of remaining packets decreases monotonically. Therefore, the probability of failure is $O(1/n^{0.2})$ in any subsequent window, and the probability of failing over all of the next 5 windows is less than $O(1/n)$. It follows that at most 6 windows are needed for all packets to succeed. ◀

4 Inductive Arguments

We present two inductive arguments for establishing upper bounds on m_i. Later in Section 5, these results are leveraged in our makespan analysis, and extracting them here allows us to modularize our presentation. Lemma 9 applies to FB, BEB, and LLB, while Lemma 10 applies to STB. We highlight that a single inductive argument would suffice for all algorithms – allowing for a simpler presentation – if we only cared about asymptotic makespan. However, in the case of FB we wish to obtain a tight bound on the first-order term, which is one of the contributions in [5].

In the following, we specify $m_0 \leq n$ since a (very) few packets may have succeeded prior to window 0; recall, this is the window where a large number of packets are expected to succeed.

▶ **Lemma 9.** *Consider a batch of $m_0 \leq n$ packets that execute over windows $w_i \geq m_0 + \sqrt{m_0}$ for all $i \geq 0$. If $m_i \geq n^{7/10}$, then $m_{i+1} \leq (4/5)\frac{m_0}{2^{2^i \lg(5/4)}}$ with error probability at most $(i+1)/n^2$.*

Proof. We argue by induction on $i \geq 0$.

Base Case. Let $i = 0$. Using Lemma 5:

$$\begin{aligned} m_1 &\leq m_0 - \frac{(1-\epsilon)m_0}{e^{m_0/(w_0-1)}} \\ &\leq m_0\left(1 - \frac{1}{e^{m_0/(w_0-1)}} + \epsilon\right) \end{aligned}$$

$$\begin{aligned} &\leq m_0\left(1 - \frac{1}{e} + \epsilon\right) \\ &\leq (0.64)m_0 \end{aligned}$$

where the last line follows by setting $\epsilon = \frac{\sqrt{4e\ln n}}{n^{1/3}}$, and assuming n is sufficiently large to satisfy the inequality; this gives an error probability of at most $1/n^2$. The base case is satisfied since $(4/5)\frac{m_0}{2^{2^i \lg(5/4)}} = (0.64)m_0$.

Induction Hypothesis (IH). For $i \geq 1$, assume $m_i \leq (4/5)\frac{m_0}{2^{2^{i-1}\lg(5/4)}}$ with error probability at most i/n^2.

Induction Step. For window $i \geq 1$, we wish to show that $m_{i+1} \leq (4/5)\frac{m_0}{2^{2^i \lg(5/4)}}$ with an error bound of $(i+1)/n^2$. Addressing the number of packets, we have:

$$\begin{aligned} m_{i+1} &\leq \frac{(5/4)m_i^2}{w_i} \\ &\leq \left(\frac{4\,m_0}{5 \cdot 2^{2^{i-1}\lg(5/4)}}\right)^2\left(\frac{5}{4w_i}\right) \\ &\leq \left(\frac{4m_0}{5 \cdot 2^{2^i \lg(5/4)}}\right)\left(\frac{m_0}{w_i}\right) \\ &< \left(\frac{4m_0}{5 \cdot 2^{2^i \lg(5/4)}}\right) \text{since } w_i > n \end{aligned}$$

The first line follows from Case 1 of Corollary 6, which we may invoke since $w_i \geq m_0 + \sqrt{m_0}$ for all $i \geq 0$, and $m_i \geq n^{7/10}$ by assumption. This yields an error of at most $1/n^2$, and so the total error is at most $i/n^2 + 1/n^2 = (i+1)/n^2$ as desired. The second line follows from the IH. ◀

A nearly identical lemma is useful for upper-bounding the makespan of STB. The main difference arises from addressing the decreasing window sizes in a run, and this necessitates the condition that $w_i \geq m_i + \sqrt{m_i}$ rather than $w_i \geq m_0 + \sqrt{m_0}$ for all $i \geq 0$. Later in Section 5, we start analyzing STB when the window size reaches $4n$; this motivates the condition that $w_i \geq 4n/2^i$ our next lemma.

▶ **Lemma 10.** *Consider a batch of $m_0 \leq n$ packets that execute over windows of size $w_i \geq m_i + \sqrt{m_i}$ and $w_i \geq 4n/2^i$ for all $i \geq 0$. If $m_i \geq n^{7/10}$, then $m_{i+1} \leq (4/5)\frac{m_0}{2^i 2^{2^i \lg(5/4)}}$ with error probability at most $(i+1)/n^2$.*

Proof. We argue by induction on $i \geq 0$.

Base Case. Nearly identical to the base case in proof of Lemma 9; note the bound on m_{i+1} is identical for $i = 0$.

Induction Hypothesis (IH). For $i \geq 1$, assume $m_i \leq (4/5)\frac{m_0}{2^{i-1}2^{2^{i-1}\lg(5/4)}}$ with error probability at most i/n^2.

Induction Step. For window $i \geq 1$, we wish to show that $m_{i+1} \leq (4/5)\frac{m_0}{2^i 2^{2^i \lg(5/4)}}$ with an error bound of $(i+1)/n^2$ (we use the same ϵ as in Lemma 9). Addressing the number of packets, we have:

$$\begin{aligned} m_{i+1} &\leq \frac{(5/4)m_i^2}{w_i} \\ &\leq \left(\frac{4m_0}{5 \cdot 2^{i-1}2^{2^{i-1}\lg(5/4)}}\right)^2 \left(\frac{5}{4w_i}\right) \\ &\leq \left(\frac{4m_0}{5 \cdot 2^i 2^{2^i \lg(5/4)}}\right)\left(\frac{m_0}{2^{i-2}w_i}\right) \\ &\leq \left(\frac{4m_0}{5 \cdot 2^i 2^{2^i \lg(5/4)}}\right) \text{since } w_i \geq 4n/2^i \end{aligned}$$

Again, the first line follows from Case 1 of Corollary 6, which we may invoke since $w_i \geq m_0 + \sqrt{m_0}$ for all $i \geq 0$, and $m_i \geq n^{7/10}$ by assumption. This gives the desired error bound of $i/n^2 + 1/n^2 = (i+1)/n^2$. The second line follows from the IH. ◀

5 Bounding Makespan

We begin by describing the windowed backoff algorithms Fixed Backoff (FB), Binary Exponential Backoff (BEB), and Log-Log Backoff (LLB) analyzed in [5]. Recall that, in each window, a packet selects a single slot uniformly at random to send in. Therefore, we need only specify how the size of successive windows change.

FB is the simplest, with all windows having size $\Theta(n)$. The value of hidden constant does not appear to be explicitly specified in the literature, but we observe that Bender et al. [5] use $3e^3$ in their upper-bound analysis. Here, we succeed using a smaller constant; namely, any value at least $1 + 1/\sqrt{n}$.

BEB has an initial window size of 1, and each successive window doubles in size.

LLB has an initial window size of 2, and for a current window size of w_i, it executes $\lceil \lg\lg(w_i) \rceil$ windows of that size before doubling; we call these sequence of same-sized windows a ***plateau***.[4]

STB is non-monotonic and executes over a doubly-nested loop. The outer loop sets the current window size w to be double that used in the preceding outer loop and each packet selects a single slot to send in; this is like BEB. Additionally, for each such w, the inner loop executes over $\lg w$ windows of decreasing size: $w, w/2, w/4, ..., 1$; this sequence of windows is referred to as a ***run***. For each window in a run, a packet chooses a slot uniformly at random in which to send.

5.1 Analysis

The following results employ tools from the prior sections a constant number of times, and each tool has error probability either $O(\log n/n^2)$ or $O(\frac{1}{n})$. Therefore, all following theorems hold with probability at least $1 - O(1/n)$, and we omit further discussion of error.

[4] As stated by Bender et al. [5], an equivalent (in terms of makespan) specification of LLB is that $w_{i+1} = (1 + 1/\lg\lg(w_i))w_i$.

▶ **Theorem 11.** *The makespan of FB with window size at least $n + \sqrt{n}$ is at most $n \lg\lg n + O(n)$ and at least $n \lg\lg n - O(n)$.*

Proof. Since $w_i \geq n + \sqrt{n}$ for all $i \geq 0$, by Lemma 9 less than $n^{7/10}$ packets remain after $\lg\lg(n) + 1$ windows; to see this, solve for i in $(4/5)\frac{n}{2^{2^i \lg(5/4)}} = n^{0.7}$. By Lemma 8, all remaining packets succeed within 6 more windows. The corresponding number of slots is $(\lg\lg n + 7)(n + \sqrt{n}) = n \lg\lg n + O(n)$. ◀

▶ **Theorem 12.** *The makespan of BEB is at most $512n \lg n + O(n)$.*

Proof. Let W be the first window of size at least $n + \sqrt{n}$ (and less than $2(n + \sqrt{n})$). Assume no packets finish before the start of W; otherwise, this can only improve the makespan. By Lemma 9 less than $n^{7/10}$ packets remain after $\lg\lg(n) + 1$ windows. By Lemma 8 all remaining packets succeed within 6 more windows. Since W has size less than $2(n + \sqrt{n})$, the number of slots until the end of W, plus those for the $\lg\lg(n) + 7$ subsequent windows, is less than:

$$\begin{aligned} & \left(\sum_{j=0}^{\lg(2(n+\sqrt{n}))} 2^j \right) + \left(\sum_{k=1}^{\lg\lg(n)+7} 2(n + \sqrt{n})2^k \right) \\ = \quad & 512(n + \sqrt{n}) \lg n + O(n) \end{aligned}$$

by the sum of a geometric series. ◀

▶ **Theorem 13.** *The makespan of STB is $O(n)$.*

Proof. Let W be the first window of size at least $4n$. Assume no packets finish before the start of W, that is $m_0 = n$; else, this can only improve the makespan.

While $m_i \geq n^{0.7}$, our analysis examines the windows in the run starting with window W, and so $w_0 \geq 4n, w_1 \geq 2n$, etc. To invoke Lemma 10, we must ensure that the condition $w_i \geq m_i + \sqrt{m_i}$ holds in each window of this run. This holds for $i = 0$, since $w_0 = 4n \geq n + \sqrt{n}$.

For $i \geq 1$, we argue this inductively by proving $m_i \leq (5/4)^{2^{i-1}-1} \frac{n}{3^{2^{i-1}}}$. For the base case $i = 1$, Lemma 5 implies that $m_1 \leq n(1 - e^{-n/(4n-1)} + \epsilon) \leq n(1 - e^{-1/3} + \epsilon) \leq n/3$, where ϵ is given in Lemma 6. For the inductive step, assume that $m_i \leq (5/4)^{2^{i-1}-1} \frac{n}{3^{2^{i-1}}}$ for all $i \geq 2$. Then, by Case 1 of Corollary 6:

$$\begin{aligned} m_{i+1} \quad & \leq \quad (5/4)m_i^2/n \\ & \leq \quad (5/4)\left((5/4)^{2^{i-1}-1} \frac{n}{3^{2^{i-1}}}\right)^2 /n \\ & \leq \quad (5/4)^{2^i-1} \frac{n}{3^{2^i}} \end{aligned}$$

where the second line follows from the assumption, and so the inductive step holds. On the other hand, at window i, $w_i \geq \frac{4n}{2^i} > \frac{4n}{(5/2)\cdot(12/5)^{2^{i-1}}} = 2 \cdot (5/4)^{2^{i-1}-1} \frac{n}{3^{2^{i-1}}} \geq 2m_i > m_i + \sqrt{m_i}$ holds.

Lemma 10 implies that after $\lg\lg n + O(1)$ windows in this run, less than $n^{0.7}$ packets remain. Pessimistically, assume no other packets finish in the run. The next run starts with a window of size at least $8n$, and by Lemma 8, all remaining packets succeed within the first 6 windows of this run.

We have shown that STB terminates within at most $\lceil \lg(n) \rceil + O(1)$ runs. The total number of slots over all of these runs is $O(n)$ by a geometric series. ◀

It is worth noting that STB has asymptotically-optimal makespan since we cannot hope to finish n packets in $o(n)$ slots.

Bender et al. [5] show that the optimal makespan for any *monotonic* windowed backoff algorithm is $O(n \lg\lg n/\lg\lg\lg n)$ and that LLB achieves this. We re-derive the makespan for LLB.

▶ **Theorem 14.** *The makespan of LLB is* $O\left(\frac{n \lg\lg n}{\lg\lg\lg n}\right)$.

Proof. For the first part of our analysis, assume $n/\ln\ln\ln n \leq m_0 \leq n$ packets remain. Consider the first window with size $w_0 = cn/\ln\ln\ln n$ for some constant $c \geq 8$. By Lemma 5, <u>each</u> window finishes at least the following number of packets:

$$\begin{aligned}
\frac{(1-\epsilon)m_0}{e^{\frac{m_0}{(cn/\ln\ln\ln n)-1}}} &> \frac{(1-\epsilon)n}{e^{\frac{n}{(cn/\ln\ln\ln n)-1}} \cdot \ln\ln\ln n} \\
&= \frac{(1-\epsilon)n}{(\ln\ln n)^{\frac{2}{c}} \cdot \ln\ln\ln n} \\
&= \frac{(1-\epsilon)n}{(\ln\ln n)^{\frac{\ln\ln\ln\ln n}{\ln\ln\ln n}+\frac{2}{c}}} \\
&> \frac{n}{(\ln\ln n)^{\frac{3}{c}}}
\end{aligned}$$

where the third line follows from noting that $(\ln\ln n)^{\ln(\ln\ln\ln n)} = (\ln\ln\ln n)^{\ln(\ln\ln n)}$, and the last line follows for sufficiently-large n. Setting $\epsilon = \sqrt{\frac{4e\ln^2(n)}{n}}$ suffices to give an error probability at most $\exp(-\frac{4e\ln^2(n)}{n} \cdot \frac{n}{2\ln\ln\ln(n)e^{\frac{n}{(cn/\ln\ln\ln n)-1}}}) \leq 1/n^2$.

Observe that in this first part of the analysis, we rely on $w_i \leq m_i - \sqrt{m_i}$ or $w_i \geq m_i + \sqrt{m_i}$ in order to apply Lemma 5. However, after enough packets succeed, neither of these inequalities may hold. But there will be at most a single plateau with windows of size $O(n/\ln\ln\ln n)$ where this occurs, since the window size will then double. During this plateau, which consists of $O(\lg\lg(n/\ln\ln\ln n)) = O(\lg\lg n)$ windows, we pessimistically assume no packets succeed.

Therefore, starting with n packets, after at most $\frac{n - n/\ln\ln\ln n}{n/(\ln\ln n)^{3/c}} + O(\lg\lg n) = O(\ln\ln n)$ windows, the number of remaining packets is less than $n/\ln\ln\ln n$, and the first part of our analysis is over.

Over the next two plateaus, LLB has at least $2\lg\lg(n) - O(1)$ windows of size $\Theta(n/\ln\ln\ln n)$. Since in this part of the analysis, $w_i \geq 8n/\ln\ln\ln n$ and $m_i < n/\ln\ln\ln n$, we have $w_i \geq m_i + \sqrt{m_i}$. Therefore, we may invoke Lemma 9, which implies that after at most $\lg\lg(n) + 1$ windows, less than $n^{0.7}$ packets remain. If at least $n^{2/5}$ packets still remain, by Case 2 of Corollary 1, at most $O(n^{2/5})$ packets remain by the end of the next window, and they will finish within an additional 6 windows by Lemma 8.

Finally, tallying up over both parts of the analysis, the makespan is $O(\ln\ln n)O(\frac{n}{\ln\ln\ln n}) = O(\frac{n\ln\ln n}{\ln\ln\ln n})$. ◀

6 Discussion

We have argued that standard Chernoff bounds can be applied to analyze singletons, and we illustrate how they simplify the analysis of several backoff algorithms under batched arrivals.

While our goal was only to demonstrate the benefits of this approach, natural extensions include the following. First, there is some slack in our arguments, and we can likely derive tighter constants in our analysis. For example, the number of windows required in Lemma 8 might be reduced; this would reduce the leading constant for our BEB analysis.

Second, we strongly believe that lower bounds can be proved using this approach. In fact, Max bets Qian (under penalty of eating bitter melon) that a lower bound on FB of $n \lg\lg n - O(n)$ can be proved, which is tight in the highest-order term.

Third, a similar treatment is possible for polynomial backoff or generalized exponential backoff (see [5] for the specification of these algorithms).

Fourth, a plausible next step is to examine whether we can extend this type of analysis to the case where packets have different sizes, as examined in [6].

References

1 Lakshmi Anantharamu, Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Medium access control for adversarial channels with jamming. In *Proceedings of the* 18th *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 89–100, 2011.

2 Antonio Fernández Anta, Miguel A. Mosteiro, and Jorge Ramón Muñoz. Unbounded contention resolution in multiple-access channels. *Algorithmica*, 67(3):295–314, 2013.

3 Baruch Awerbuch, Andrea Richa, and Christian Scheideler. A jamming-resistant MAC protocol for single-hop wireless networks. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 45–54, 2008.

4 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, September 1999.

5 Michael A. Bender, Martin Farach-Colton, Simai He, Bradley C. Kuszmaul, and Charles E. Leiserson. Adversarial contention resolution for simple channels. In *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 325–332, 2005.

6 Michael A. Bender, Jeremy T. Fineman, and Seth Gilbert. Contention Resolution with Heterogeneous Job Sizes. In *Proceedings of the 14th Conference on Annual European Symposium (ESA)*, pages 112–123, 2006.

7 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. How to scale exponential backoff: Constant throughput, polylog access attempts, and robustness. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016.

8 Michael A. Bender, Tsvi Kopelowitz, Seth Pettie, and Maxwell Young. Contention resolution with log-logstar channel accesses. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 499–508, 2016.

9 Petra Berenbrink, Artur Czumaj, Matthias Englert, Tom Friedetzky, and Lars Nagel. Multiple-choice balanced allocation in (almost) parallel. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX-RANDOM)*, pages 411–422, 2012.

10 Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.

11 Petra Berenbrink, Kamyar Khodamoradi, Thomas Sauerwald, and Alexandre Stauffer. Balls-into-bins with nearly optimal load distribution. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 326–335, 2013.

12 Yi-Jun Chang, Varsha Dani, Thomas P. Hayes, Qizheng He, Wenzheng Li, and Seth Pettie. The energy complexity of broadcast. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 95–104, 2018.

13 Yi-Jun Chang, Tsvi Kopelowitz, Seth Pettie, Ruosong Wang, and Wei Zhan. Exponential separations in the energy complexity of leader election. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 771–783, 2017.

14 Bogdan S. Chlebus, Gianluca De Marco, and Dariusz R. Kowalski. Scalable wake-up of multi-channel single-hop radio networks. *Theoretical Computer Science*, 615(C):23–44, February 2016.

15 Bogdan S. Chlebus, Leszek Gasieniec, Dariusz R. Kowalski, and Tomasz Radzik. On the wake-up problem in radio networks. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 347–359, 2005.

16 Bogdan S. Chlebus and Dariusz R. Kowalski. A better wake-up in radio networks. In *Proceedings of 23rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 266–274, 2004.

17 Marek Chrobak, Leszek Gasieniec, and Dariusz R. Kowalski. The wake-up problem in multihop radio networks. *SIAM Journal on Computing*, 36(5):1453–1471, 2007.

18 Richard Cole, Alan M. Frieze, Bruce M. Maggs, Michael Mitzenmacher, Andréa W. Richa, Ramesh K. Sitaraman, and Eli Upfal. On balls and bins with deletions. In *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 145–158, 1998.

19 A. Czumaj and V. Stemann. Randomized Allocation Processes. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 194–203, 1997.

20 Gianluca De Marco and Grzegorz Stachowiak. Asynchronous shared channel. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 391–400, 2017.

21 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 1st edition, 2009.

22 Devdatt Dubhashi and Desh Ranjan. Balls and Bins: A Study in Negative Dependence. *Random Structures & Algorithms*, 13(2):99–124, 1998. `doi:10.1002/(SICI)1098-2418(199809)13:2<99::AID-RSA1>3.0.CO;2-M`.

23 Jeremy T. Fineman, Seth Gilbert, Fabian Kuhn, and Calvin Newport. Contention resolution on a fading channel. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 155–164, 2016.

24 Jeremy T. Fineman, Calvin Newport, and Tonghe Wang. Contention resolution on multiple channels with collision detection. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 175–184, 2016.

25 Mihály Geréb-Graus and Thanasis Tsantilas. Efficient optical communication in parallel computers. In *Proceedings 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 41–48, 1992.

26 Leslie Ann Goldberg and Philip D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. *Journal of Computer and System Sciences*, 58(1):232–258, 1999. `doi:10.1006/jcss.1998.1590`.

27 Leslie Ann Goldberg, Philip D. Mackenzie, Mike Paterson, and Aravind Srinivasan. Contention resolution with constant expected delay. *Journal of the ACM*, 47(6):1048–1096, 2000.

28 Jonathan Goodman, Albert G. Greenberg, Neal Madras, and Peter March. Stability of binary exponential backoff. *Journal of the ACM*, 35(3):579–602, July 1988.

29 Ronald I. Greenberg and Charles E. Leiserson. Randomized routing on fat-trees. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 241–249, 1985.

30 Johan Hastad, Tom Leighton, and Brian Rogoff. Analysis of backoff protocols for multiple access channels. *SIAM Journal on Computing*, 25(4):1996, 740-774.

31 IEEE. IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks – Specific requirements - Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016.

32 Tomasz Jurdzinski and Grzegorz Stachowiak. The cost of synchronizing multiple-access channels. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 421–430, 2015.

33 A R Karlin and E Upfal. Parallel Hashing - An Efficient Implementation of Shared Memory. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 160–168, 1986.

34 James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach*. Pearson, 6th edition, 2013.

35 Christoph Lenzen and Roger Wattenhofer. Tight Bounds for Parallel Randomized Load Balancing: Extended Abstract. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 11–20, 2011.

36 Gianluca De Marco and Dariusz R. Kowalski. Fast nonadaptive deterministic algorithm for conflict resolution in a dynamic multiple-access channel. *SIAM Journal on Computing*, 44(3):868–888, 2015.

37 Gianluca De Marco and Dariusz R. Kowalski. Contention resolution in a non-synchronized multiple access channel. *Theoretical Computer Science*, 689:1–13, 2017.

38 Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

39 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

40 Michael David Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, 1996.

41 K. Nakano and S. Olariu. Uniform leader election protocols for radio networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):516–526, May 2002. `doi:10.1109/TPDS.2002.1003864`.

42 Adrian Ogierman, Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Sade: competitive MAC under adversarial SINR. *Distributed Computing*, 31(3):241–254, June 2018.

43 Prabhakar Raghavan and Eli Upfal. Stochastic contention resolution with short delays, April 1999.

44 Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. A jamming-resistant MAC protocol for multi-hop wireless networks. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 179–193, 2010.

45 Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive and fair medium access despite reactive jamming. In *Proceedings of the* 31^{st} *International Conference on Distributed Computing Systems (ICDCS)*, pages 507–516, 2011.

46 Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive and Fair Throughput for Co-existing Networks Under Adversarial Interference. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 291–300, 2012.

47 Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

48 X. Sun and L. Dai. Backoff Design for IEEE 802.11 DCF Networks: Fundamental Tradeoff and Design Criterion. *IEEE/ACM Transactions on Networking*, 23(1):300–316, 2015.

49 Eli Upfal. Efficient Schemes for Parallel Communication. *J. ACM*, 31(3):507–517, June 1984.

50 Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003.

51 Dan E. Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM J. Comput.*, 15(2):468–477, May 1986.

52 D. Yin, K. Lee, R. Pedarsani, and K. Ramchandran. Fast and Robust Compressive Phase Retrieval with Sparse-Graph Codes. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 2583–2587, June 2015.

Appendix

A Chernoff Bounds and Property 1

In Problem 1.8 of Dubhashi and Panconesi [21], the following question is posed: Show that if Property 1 holds, then Theorem 3 holds. We are invoking this result, but an argument is absent in [21].

We bridge this gap with Claim 15 below. This fits directly into the derivation of Chernoff bounds given in Dubhashi and Panconesi [21]. In particular, the line above Equation 1.3 on page 4 of [21] claims equality for Equation 10 below by invoking independence of the random variables. Here, Claim 15 gives an inequality (in the correct direction) and the remainder of the derivation in [21] follows without any further modifications.

▷ Claim 15. Let $X_1, \cdots, X_n$ be a set of indicator random variables satisfying the property:

$$Pr\left[\bigwedge_{i\in\mathbb{S}} X_i = 1\right] \leq \prod_{i\in\mathbb{S}} Pr\left[X_i = 1\right] \tag{9}$$

for all subsets $\mathbb{S} \subset \{1, \cdots, n\}$. Then the following holds:

$$E\left[\prod_{i=1}^{n} e^{\lambda X_i}\right] \leq \prod_{i=1}^{n} E\left[e^{\lambda X_i}\right] \tag{10}$$

Proof. Let $\mathbb{N}$ denote the set of strictly positive integers. First, we need to point out two properties of indicator random variables

(i) $X_i^k = X_i$ for all $k \in \mathbb{N}$; and

(ii) $E\left[X_i\right] = Pr\left[X_i = 1\right]$, and $E\left[\prod_{i\in\mathbb{S}} X_i\right] = Pr\left[\bigwedge_{i\in\mathbb{S}} X_i = 1\right]$ for all subset $\mathbb{S}$.

By Taylor expansion we have $e^{\lambda X_i} = \sum_{k=0}^{\infty} \lambda^k \frac{X_i^k}{k!}$, and then,

$$E\left[e^{\lambda X_i}\right] = \sum_{k=0}^{\infty} \lambda^k \frac{E\left[X_i^k\right]}{k!} \tag{11}$$

Thus, the product in the left hand side (LHS) of (10) becomes $\prod_{i=1}^{n} e^{\lambda X_i} = \prod_{i=1}^{n}\left(\sum_{k=0}^{\infty} \frac{\lambda^k}{k!} X_i^k\right)$, which can be written as a polynomial function of λ, i.e. $\sum_{r=0}^{\infty} f_r \lambda^r$, where f_r are coefficients which may contain the indicator random variables X_is. Here $f_0 = 1$. To get the expression of f_r for $r \geq 1$, we first define a set, for all integers $k, r \in \mathbb{N}$ with $k \leq r$, let $\mathcal{I}(k, r) = \{(d_1, d_2, \cdots, d_k) : d_1, \cdots, d_k \in \mathbb{N}, d_1 \leq d_2 \leq \cdots \leq d_k, d_1 + d_2 + \cdots + d_k = r\}$. Then the coefficients f_r, $r \geq 1$, can be expressed as

$$f_r = \sum_{k=1}^{\min\{r,n\}} \sum_{(d_1,\cdots,d_k)\in\mathcal{I}(r,k)} \sum_{1\leq i_1\neq i_2\neq\cdots\neq i_k\leq n} \frac{X_{i_1}^{d_1}}{d_1!}\frac{X_{i_2}^{d_2}}{d_2!}\cdots\frac{X_{i_k}^{d_k}}{d_k!}. \tag{12}$$

For example,

$$
\begin{aligned}
f_1 &= \sum_{i=1}^{n} X_i \\
f_2 &= \sum_{i=1}^{n} \frac{X_i^2}{2!} + \sum_{1 \le i_1 \ne i_2 \le n} X_{i_1} X_{i_2} \\
f_3 &= \sum_{i=1}^{n} \frac{X_i^3}{3!} + \sum_{1 \le i_1 \ne i_2 \le n} X_{i_1} \frac{X_{i_2}^2}{2!} + \sum_{1 \le i_1 \ne i_2 \ne n_3 \le n} X_{i_1} X_{i_2} X_{i_3} \\
&\vdots
\end{aligned}
$$

With the expression (12), the LHS becomes

$$
\begin{aligned}
\text{LHS} &= 1 + \sum_{r=1}^{\infty} \lambda^r \sum_{k=1}^{\min\{r,n\}} \sum_{(d_1,\cdots,d_k)\in\mathcal{I}(r,k)} \sum_{1\le i_1 \ne i_2 \ne \cdots \ne i_k \le n} E\left[\frac{X_{i_1}^{d_1}}{d_1!}\frac{X_{i_2}^{d_2}}{d_2!}\cdots\frac{X_{i_k}^{d_k}}{d_k!}\right] \\
&= 1 + \sum_{r=1}^{\infty} \lambda^r \sum_{k=1}^{\min\{r,n\}} \sum_{(d_1,\cdots,d_k)\in\mathcal{I}(r,k)} \sum_{1\le i_1 \ne i_2 \ne \cdots \ne i_k \le n} \frac{E\left[X_{i_1}^{d_1}X_{i_2}^{d_2}\cdots X_{i_k}^{d_k}\right]}{d_1!d_2!\cdots d_k!}
\end{aligned}
$$

Similarly, with the Taylor expansion of (11), the product in the right hand side (RHS) of (10) becomes

$$
\begin{aligned}
RHS &= \prod_{i=1}^{n}\left(\sum_{k=0}^{\infty}\lambda^k \frac{E\left[X_i^k\right]}{k!}\right) \\
&= 1 + \sum_{r=1}^{\infty} \lambda^r \sum_{k=1}^{\min\{r,n\}} \sum_{(d_1,\cdots,d_k)\in\mathcal{I}(r,k)} \sum_{1\le i_1 \ne i_2 \ne \cdots \ne i_k \le n} \frac{E\left[X_{i_1}^{d_1}\right]}{d_1!}\frac{E\left[X_{i_2}^{d_2}\right]}{d_2!}\cdots\frac{E\left[X_{i_k}^{d_k}\right]}{d_k!} \\
&= 1 + \sum_{r=1}^{\infty} \lambda^r \sum_{k=1}^{\min\{r,n\}} \sum_{(d_1,\cdots,d_k)\in\mathcal{I}(r,k)} \sum_{1\le i_1 \ne i_2 \ne \cdots \ne i_k \le n} \frac{E\left[X_{i_1}^{d_1}\right]E\left[X_{i_2}^{d_2}\right]\cdots E\left[X_{i_k}^{d_k}\right]}{d_1!d_2!\cdots d_k!}
\end{aligned}
$$

By the above-mentioned two properties (i) and (ii) of indicator random variables, then

$$
\begin{aligned}
E\left[X_{i_1}^{d_1}X_{i_2}^{d_2}\cdots X_{i_k}^{d_k}\right] &= E\left[X_{i_1}X_{i_2}\cdots X_{i_k}\right] = Pr\left[X_{i_1}=1, X_{i_2}=1,\cdots,X_{i_k}=1\right] \\
E\left[X_{i_1}^{d_1}\right]E\left[X_{i_2}^{d_2}\right]\cdots E\left[X_{i_k}^{d_k}\right] &= E\left[X_{i_1}\right]E\left[X_{i_2}\right]\cdots E\left[X_{i_k}\right] \\
&= Pr\left[X_{i_1}=1\right]Pr\left[X_{i_2}=1\right]\cdots Pr\left[X_{i_k}=1\right].
\end{aligned}
$$

By the condition (9), we have $Pr\left[X_{i_1}=1, X_{i_2}=1,\cdots,X_{i_k}=1\right] \le Pr\left[X_{i_1}=1\right]Pr\left[X_{i_2}=1\right]\cdots Pr\left[X_{i_k}=1\right]$, and thus

$$
E\left[X_{i_1}^{d_1}X_{i_2}^{d_2}\cdots X_{i_k}^{d_k}\right] \le E\left[X_{i_1}^{d_1}\right]E\left[X_{i_2}^{d_2}\right]\cdots E\left[X_{i_k}^{d_k}\right].
$$

Thus (10) holds. ◁