# 2nd Symposium on Algorithmic Foundations of Dynamic Networks

**SAND 2023, June 19–21, 2023, Pisa, Italy**

Edited by

David Doty

Paul Spirakis


LIPIcs – Vol. 257 – SAND 2023 www.dagstuhl.de/lipics

*Editors*

**David Doty**
Department of Computer Science, University of California, Davis, CA, USA
doty@ucdavis.edu

**Paul Spirakis**
University of Liverpool, UK
P.Spirakis@liverpool.ac.uk

*ACM Classification 2012*
Theory of computation; Mathematics of computing; Networks

**ISBN 978-3-95977-275-4**

*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/978-3-95977-275-4.

*Publication date*
June, 2023

*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.

*License*
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.SAND.2023.0

**ISBN 978-3-95977-275-4** **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.

*Editorial Board*

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB and Nanyang Technological University, SG)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

**ISSN 1868-8969**

**https://www.dagstuhl.de/lipics**

# Contents

Preface
*David Doty and Paul Spirakis* ..... 0:vii

List of Authors
..... 0:ix

## Papers

Snapshot Disjointness in Temporal Graphs
*Allen Ibiapina and Ana Silva* ..... 1:1–1:20

Partial Gathering of Mobile Agents in Dynamic Tori
*Masahiro Shibata, Naoki Kitamura, Ryota Eguchi, Yuichi Sudo, Junya Nakamura, and Yonghwan Kim* ..... 2:1–2:22

Bond Percolation in Small-World Graphs with Power-Law Distribution
*Luca Becchetti, Andrea Clementi, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi* ..... 3:1–3:22

Computing Temporal Reachability Under Waiting-Time Constraints in Linear Time
*Filippo Brunelli and Laurent Viennot* ..... 4:1–4:11

Complexity of Motion Planning of Arbitrarily Many Robots: Gadgets, Petri Nets, and Counter Machines
*Joshua Ani, Michael Coulombe, Erik D. Demaine, Yevhenii Diomidov, Timothy Gomez, Dylan Hendrickson, and Jayson Lynch* ..... 5:1–5:21

Adaptive Collective Responses to Local Stimuli in Anonymous Dynamic Networks
*Shunhao Oh, Dana Randall, and Andréa W. Richa* ..... 6:1–6:23

When Should You Wait Before Updating? – Toward a Robustness Refinement
*Swan Dubois, Laurent Feuilloley, Franck Petit, and Mikaël Rabie* ..... 7:1–7:15

Dynamic Graphs Generators Analysis: An Illustrative Case Study
*Vincent Bridonneau, Frédéric Guinand, and Yoann Pigné* ..... 8:1–8:19

Complexity of the Temporal Shortest Path Interdiction Problem
*Jan Boeckmann, Clemens Thielen, and Alina Wittmann* ..... 9:1–9:20

A Connectivity-Sensitive Approach to Consensus Dynamics
*Bernard Chazelle and Kritkorn Karntikoon* ..... 10:1–10:17

Making Self-Stabilizing Algorithms for Any Locally Greedy Problem
*Johanne Cohen, Laurence Pilard, Mikaël Rabie, and Jonas Sénizergues* ..... 11:1–11:17

Covert Computation in the Abstract Tile-Assembly Model
*Robert M. Alaniz, David Caballero, Timothy Gomez, Elise Grizzell, Andrew Rodriguez, Robert Schweller, and Tim Wylie* ..... 12:1–12:17

2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023).
Editors: David Doty and Paul Spirakis
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Restless Exploration of Periodic Temporal Graphs
*Thomas Bellitto, Cyril Conchon–Kerjan, and Bruno Escoffier* ..... 13:1–13:15

Multistage Shortest Path: Instances and Practical Evaluation
*Markus Chimani and Niklas Troost* ..... 14:1–14:19

# Preface

This volume contains the papers that were presented at the 2nd Symposium on Algorithmic Foundations of Dynamic Networks, held in Pisa, Italy, June 19–21, 2023.

The Symposium on Algorithmic Foundations of Dynamic Networks (SAND) is a new conference whose objective is to become the primary venue for original research on fundamental aspects of computing in dynamic networks and computational dynamics, bringing together researchers from computer science and related areas. SAND is seeking important contributions from all viewpoints, including theory and practice, characterized by a marked algorithmic aspect and addressing or being motivated by the role of dynamics in computing. It welcomes both conceptual and technical contributions, as well as novel ideas and new problems that will inspire the community and facilitate the further growth of the area.

The program committee of SAND 2022 consisted of

- David Doty (University of California, Davis, USA, co-chair),
- Paul Spirakis (University of Liverpool, UK, co-chair),
- Mostefaoui Achour (Université de Nantes, France),
- Hagit Attiya (Technion, Israel),
- Petra Berenbrink (Universität Hamburg, Germany),
- Silvia Bonomi (Sapienza Università di Roma, Italy),
- Janna Burman (Université Paris-Saclay, France),
- Arnaud Casteigts (University of Bordeaux, France),
- Ho-Lin Chen (National Taiwan University),
- Giuseppe Di Luna (Sapienza Università di Roma, Italy),
- Mahsa Eftekhari (Microsoft, USA),
- Jessica Enright (University of Glasgow, UK),
- Javier Esparza (Technische Universität München, Germany),
- Pierre Fraigniaud (CNRS, France),
- Leszek Gasieniec (University of Liverpool, UK),
- Chryssis Georgiou (University of Cyprus),
- Peter Kling (Universität Hamburg, Germany),
- Dariusz Kowalski (Augusta University, USA),
- Marios Mavronicolas (University of Cyprus),
- George Mertzios (Durham University, UK),
- Othon Michail (University of Liverpool, UK),
- Slobodan Mitrović (University of California, Davis, USA),
- Sotiris Nikoletseas (Patras University, Greece),
- Thomas Nowak (Université Paris-Saclay, France),
- Giuseppe Prencipe (University of Pisa, Italy),
- Andrea Richa (Arizona State University, USA),
- Elad Schiller (Chalmers University of Technology, Sweden),
- Stefan Schmid (Technische Universität Berlin, Germany),
- George Skretas (Hasso Plattner Institute, Germany),
- Yuichi Sudo (Hosei University, Japan),
- Przemysław Uznański (University of Wrocław, Poland),
- Hirozumi Yamaguchi (Osaka University, Japan),
- Yukiko Yamauchi (Kyushu University, Japan).

2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023).
Editors: David Doty and Paul Spirakis
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SAND 2023 received 20 submissions. The review process was double-blind and each paper was assigned to at least three members of the program committee with relevant expertise and eventually reviewed by them and/or by additional reviewers whenever needed. The program committee accepted 14 papers that cover a wide range of topics in the broad area of algorithmic foundations of dynamic networks and computational dynamics, including DNA self-assembly, dynamic networks and distributed algorithms, mobile computing and robotics, and temporal and dynamic graph algorithms. Keynote talks were given by distinguished researchers, to whom we are grateful: Fabian Kuhn (Albert-Ludwigs-Universit at), Kitty Meeks (University of Glasgow), and Nicola Santoro (Carleton University).

The program committee selected the paper "*When Should You Wait Before Updating? – Toward a Robustness Refinement*" by Swan Dubois, Laurent Feuilloley, Franck Petit and Mika el Rabie, for the Best Paper Award and the paper "*Snapshot Disjointness in Temporal Graphs*" by Allen Ibiapina and Ana Silva, for the Best Student Paper Award. We wish to thank the members of the various committees of SAND as well as its advisory board, for all the hard work that they have put and which has made it possible to set up a new conference. All have been supportive throughout. We are grateful to the program committee members and to the additional reviewers for devoting time and effort in order to come up with a strong conference program. A special thanks goes to the general chair of the organizing committee, Giuseppe Prencipe, and organizing committee members Silvia Filogna and Samuele Bonini. We are also indebted to the chair of the SAND steering committee, Paola Flocchini, for all her support, to Giuseppe Prencipe for handling all the financial aspects, and to George Skretas for helping on publicity matters.

Above all, we thank the authors for submitting their work to SAND 2023. We can assure the reader that in this volume they will find well-presented ideas and results that make substantial contributions to our knowledge on the role of dynamics in computing. We do believe that this volume will inspire further work and will contribute to the further growth of this exciting research area.

June, 2023 David Doty and Paul Spirakis

# List of Authors


Robert M. Alaniz (12)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Joshua Ani (5)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Luca Becchetti (3)
Sapienza University of Rome, Italy

Thomas Bellitto (13)
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Jan Boeckmann (9)
TUM Campus Straubing for Biotechnology and Sustainability, Hochschule Weihenstephan-Triesdorf, Germany

Vincent Bridonneau (8)
Université Le Havre Normandie, Normandie Univ, LITIS EA 4108, 76600 Le Havre, France

Filippo Brunelli (4)
Université Paris Cité, Inria, CNRS, IRIF, Paris, France

David Caballero (12)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Bernard Chazelle (10)
Department of Computer Science, Princeton University, NJ, USA

Markus Chimani (14)
Theoretical Computer Science, Universität Osnabrück, Germany

Andrea Clementi (3)
University of Rome Tor Vergata, Italy

Johanne Cohen (11)
Université Paris-Saclay, CNRS, LISN, 91405, Orsay, France

Cyril Conchon-Kerjan (13)
DIENS, Ecole normale supérieure, F-75005 Paris, France

Michael Coulombe (5)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Erik D. Demaine (5)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Yevhenii Diomidov (5)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Swan Dubois (7)
Sorbonne Université, CNRS, LIP6, DELYS, Paris, France

Ryota Eguchi (2)
NAIST, Nara, Japan

Bruno Escoffier (13)
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France; Institut Universitaire de France, Paris, France

Laurent Feuilloley (7)
Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, Villeurbanne, France

Timothy Gomez (5, 12)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Elise Grizzell (12)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Frédéric Guinand (8)
Université Le Havre Normandie, Normandie Univ, LITIS EA 4108, 76600 Le Havre, France

Dylan Hendrickson (5)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Allen Ibiapina (1)
ParGO Group, Departament of Mathematics, Federal University of Ceará, Fortaleza, Brazil

Kritkorn Karntikoon (10)
Department of Computer Science, Princeton University, NJ, USA

Yonghwan Kim (2)
Nagoya Institute of Technology, Aichi, Japan



2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023).
Editors: David Doty and Paul Spirakis
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Naoki Kitamura (2)
Osaka University, Japan

Jayson Lynch (5)
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Junya Nakamura (2)
Toyohashi University of Technology, Aichi, Japan

Shunhao Oh (6)
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Francesco Pasquale (3)
University of Rome Tor Vergata, Italy

Franck Petit (7)
Sorbonne Université, CNRS, LIP6, DELYS, Paris, France

Yoann Pigné (8)
Université Le Havre Normandie, Normandie Univ, LITIS EA 4108, 76600 Le Havre, France

Laurence Pilard (11)
LI-PaRAD, UVSQ, Université Paris-Saclay, France

Mikaël Rabie (7, 11)
Université Paris Cité, CNRS, IRIF, Paris, France

Dana Randall (6)
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Andréa W. Richa (6)
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Andrew Rodriguez (12)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Robert Schweller (12)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Masahiro Shibata (2)
Kyushu Institute of Technology, Fukuoka, Japan

Ana Silva (1)
ParGO Group, Departament of Mathematics, Federal University of Ceará, Fortaleza, Brazil

Yuichi Sudo (2)
Hosei University, Tokyo, Japan

Jonas Sénizergues (11)
Université Paris-Saclay, CNRS, LISN, 91405, Orsay, France

Clemens Thielen (9)
TUM Campus Straubing for Biotechnology and Sustainability, Hochschule Weihenstephan-Triesdorf, Germany; Department of Mathematics, School of Computation, Information and Technology, Technische Universität München, Germany

Luca Trevisan (3)
Bocconi University, Milan, Italy

Niklas Troost (14)
Theoretical Computer Science, Universität Osnabrück, Germany

Laurent Viennot (4)
Université Paris Cité, Inria, CNRS, IRIF, Paris, France

Alina Wittmann (9)
Department of Mathematics, School of Computation, Information and Technology, Technische Universität München, Germany

Tim Wylie (12)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Isabella Ziccardi (3)
Bocconi University, Milan, Italy

# Snapshot Disjointness in Temporal Graphs

**Allen Ibiapina**
ParGO Group, Departament of Mathematics, Federal University of Ceará, Fortaleza, Brazil

**Ana Silva**[1]
ParGO Group, Departament of Mathematics, Federal University of Ceará, Fortaleza, Brazil

## Abstract

In the study of temporal graphs, only paths respecting the flow of time are relevant. In this context, many concepts of walks disjointness have been proposed over the years, and the validity of Menger's Theorem, as well as the complexity of related problems, has been investigated. Menger's Theorem states that the maximum number of disjoint paths between two vertices is equal to the minimum number of vertices required to disconnect them. In this paper, we introduce and investigate a type of disjointness that is only time dependent. Two walks are said to be *snapshot disjoint* if they are not active in a same snapshot (also called timestep). The related paths and cut problems are then defined and proved to be W[1]-hard and XP-time solvable when parameterized by the size of the solution. Additionally, in the light of the definition of Mengerian graphs given by Kempe, Kleinberg and Kumar in their seminal paper (STOC'2000), we define a *Mengerian graph for time* as a graph $G$ for which there is no time labeling for its edges where Menger's Theorem does not hold in the context of snapshot disjointness. We then give a characterization of Mengerian graphs in terms of forbidden structures and provide a polynomial-time recognition algorithm. Finally, we also prove that, given a temporal graph $(G, \lambda)$ and a pair of vertices $s, z \in V(G)$, deciding whether at most $h$ multiedges can separate $s$ from $z$ is NP-complete, where one multiedge $uv$ is the set of all edges with endpoints $u$ and $v$.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorics; Mathematics of computing → Paths and connectivity problems

**Keywords and phrases** Temporal graphs, Menger's Theorem, Snapshot disjointness

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.1

**Funding** This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, CNPq grants 303803/2020-7, and FUNCAP/CNPq PNE-0112-00061.01.00/16 and MLC-0191-00056.01.00/22.

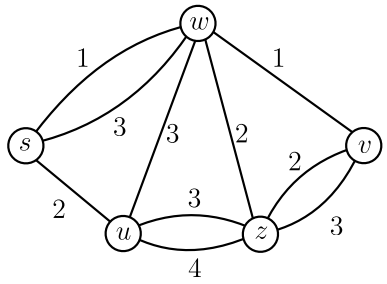
## 1 Introduction

A temporal graph can be described as a graph that varies in time. Such objects can be modeled in different ways, usually according to the application being considered, and have appeared in the literature under many names as for instance dynamic networks [19], temporal networks [9], time-varying graphs [4], etc. For surveys we refer the reader to [9, 14]. In this paper, we consider a *temporal graph* to be a pair $(G, \lambda)$, where $G$ is a multigraph (henceforward called just graph) and $\lambda$ is a function, called *timefunction*, that relates each edge to a discrete label telling when such edge is *active*. The value $\max_{e \in E(G)} \lambda(e)$ is called *lifetime* and is denoted by $\tau$. Also, graph $G$ is called the *base graph*. See Figure 1a for an example.

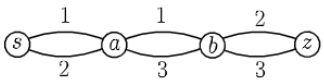
Many practical problems are modeled as temporal graphs (see [9] for a nice collection of practical examples), and among the most common ones are those related to temporal walks and connectivity. A *temporal walk* is a walk that respects the flow of time; for simplicity, we

[1] This work was partially developed during this author's stay as visiting professor at the University of Florence.


© Allen Ibiapina and Ana Silva;
licensed under Creative Commons License CC-BY 4.0




**(a)** Temporal graph with lifetime 4.




**(b)** Pair $s, z$ whose cut must be bigger than the number of snapshot disjoint paths.

**Figure 1** Examples used throughout the text to clarify some concepts.

represent temporal walks as sequences of vertices and timesteps. For instance, in Figure 1a, the sequence $(w, 2, z, 2, v, 3, z, 4, u)$ is a temporal walk between $w$ and $u$ (also called temporal $w, u$-walk). A *temporal path* is then defined as a temporal walk whose vertices are not repeated. Hence, the previously mentioned temporal $w, u$-walk is not a path, while the walk $(w, 2, z, 4, u)$ is a temporal path. Additionally, some authors deal only with walks and paths whose edges are active in strictly increasing times; in such case, $(s, 1, w, 2, z, 3, v)$ is a valid temporal $s, v$-path, while $(s, 2, u, 3, z, 3, v)$ is not. To distinguish from these, we say that a walk/path is *strict* if the edges are active in strictly increasing times, and that it is *non-strict* if they are active in non-decreasing times.

In contrast with classic graph theory, when dealing with walks and paths in temporal graphs, sometimes the connectivity problems defined on walks differ from those defined on paths. See for instance [1, 5, 7, 10]. This is not the case for the problems investigated here, and this is why we interchangeably use the terms walks and paths.

Connectivity problems concern the *robustness* of a network, which translates into knowing how many independent (or disjoint) ways there are to go from one vertex to another, and how easy it is to break such connections. In this paper, we introduce a new robustness concept that relies on the time aspect of a network. To better understand these concepts, consider the following scenario. Suppose a temporal graph $(G, \lambda)$ models a communication network. Such network might be prone to interruptions of all communications at a given timestep due to attacks, blackouts, maintenance, etc. A good measure of robustness of such networks could then be the minimum number of timesteps in which the communications must get interrupted in order to break all possible connections between a pair of vertices. A network with higher measure means that it is less susceptible to failing under such interruptions and hence is considered more robust. In Figure 1a, for instance, if there is an interruption on timesteps 2 and 3, then vertex $s$ cannot relay a message to $z$ anymore, while it still can relay messages to $v$ through the path $(z, 1, w, 1, v)$.

To model such scenario, we say that two temporal $s, z$-paths $P$ and $Q$ are *snapshot disjoint* if, at any given timestep, at most one between $P$ and $Q$ is traversing any edges. For example, in Figure 1a, paths $(s, 1, w, 1, v, 2, z)$ and $(s, 3, w, 3, u, 3, z)$ are two snapshot disjoint temporal $s, z$-paths. We also say that a set $S$ of timesteps is a *snapshot $s, z$-cut* if every temporal $s, z$-path uses an edge active in timestep $i$ for some $i \in S$. For example, in Figure 1a, $S = \{2, 3\}$ is a snapshot $s, z$-cut. The following problems are then defined.

$\leq h$-SNAPSHOT $s, z$-CUT
**Input.** A temporal graph $(G, \lambda)$, vertices $s, z \in V(G)$, and an integer $h$.
**Question.** Is there a snapshot $s, z$-cut in $(G, \lambda)$ of size at most $h$?

$\geq k$-SNAPSHOT DISJOINT TEMPORAL $s, z$-PATHS
**Input.** A temporal graph $(G, \lambda)$, a pair of vertices $s, z \in V(G)$, and a positive integer $k$.
**Question.** Is there a set of snapshot disjoint temporal $s, z$-paths in $(G, \lambda)$ of size at least $k$?

We prove that, when parameterized by $h$ and $k$ respectively, both problems are W[1]-hard, and that this is best possible, i.e., that they are also XP. While the XP algorithm for SNAPSHOT $s, z$-CUT follows easily from the definition and the fact that we can test all possible cuts in XP time (namely, $O(\tau^h)$ time), the algorithm for SNAPSHOT DISJOINT TEMPORAL $s, z$-PATHS is much more involved and uses a technique similar to the one applied to find $k$ vertex disjoint paths between $k$ given pairs of vertices (also known as the $k$-LINKAGE PROBLEM) in a DAG [17]. As we will see in the related works, this is the first result of this kind, with all previously defined disjointness either having the related paths problem polynomial-time solvable or para-NP-complete (i.e., NP-complete for fixed values of $k$).

A celebrated result in classic graph theory tells us that, in a graph $G$ and for every pair $s, z \in V(G)$, the maximum number of internally vertex disjoint $s, z$-paths is equal to the minimum size of an $s, z$-cut (vertices whose removal breaks all $s, z$-paths). This is the well known Menger's Theorem, and it holds on both undirected and directed graphs, as well as for edge-disjoint paths and edge cuts. When translating these concepts to temporal graphs, it is natural to ask whether a version of Menger's Theorem holds. The answer in our context is no, as can be witnessed by the example in Figure 1b. Note that any two temporal $s, z$-paths intersect in some timestep, while there is no snapshot $s, z$-cut of size 1. Indeed, $(s, 1, a, 1, b, 2, z)$ does not use edges active in timestep 3, $(s, 2, a, 3, b, 3, z)$ does not use 1, and $(s, 1, a, 1, b, 3, z)$ does not use 2.

In their seminal paper, Kempe, Kleinberg and Kumar [13], in the context of vertex disjoint temporal paths, defined a Mengerian graph as being a graph where Menger's Theorem would hold for every choice of timefunction. They then characterize these graphs when constrained to simple graphs (every multiedge has multiplicity 1), and more recently their result was generalized to allow for multigraphs [11]. Here, we say that $G$ is *Mengerian for time* if, for every timefunction $\lambda$ and every $s, z \in V(G)$, the maximum number of snapshot disjoint temporal $s, z$-paths in $(G, \lambda)$ is equal to the minimum size of a snapshot $s, z$-cut. In other words, the snapshot disjoint version of Menger's Theorem always holds on temporal graphs whose base graph is $G$. We then give the following characterization, which will be proved in Section 5. The formal definition of an m-topological minor is given in Section 2, but for now it suffices to say that it is a generalization of topological minors that preserves the multiplicity of the multiedges.

▶ **Theorem 1.** *Let $G$ be a graph. Then $G$ is Mengerian for time if and only if $G$ does not have any of the graphs in Figure 2 as m-topological minor. Moreover, we can recognize whether $G$ is Mengerian for time in polynomial time.*

Finally, in order to fill an open entry related to multiedge disjoint temporal paths (the definition is presented shortly), we prove in Section 6 that the related cut problem is NP-complete even if the temporal graph has lifetime equal to 2.

## Related problems

As snapshot disjointness is a newly introduced concept, no previous results exist. We then refer the reader to the many results about Menger's related concepts in temporal graphs. In this context, the vertex disjoint version of Menger's Theorem was proved not to hold by Berman [3]. Since then a number of papers have investigated the complexity of

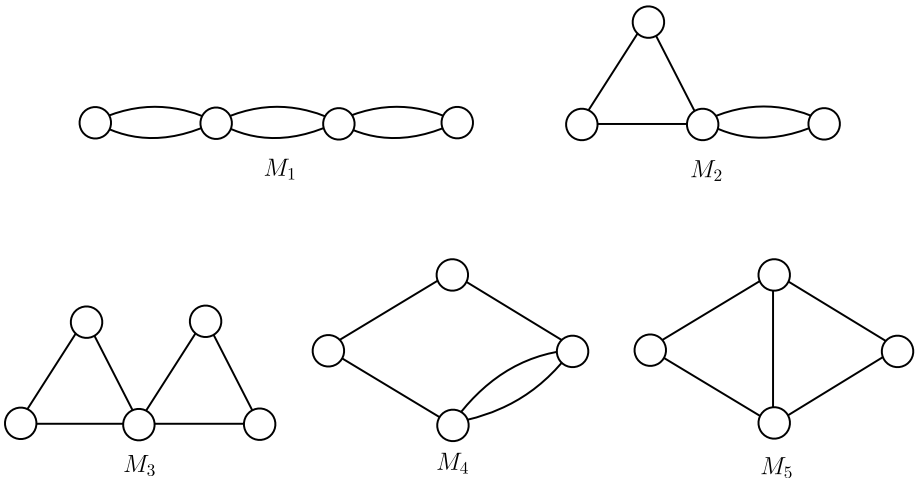



**Figure 2** Graphs in the set $\mathcal{M}$.

related problems [13, 20], as well as new structural concepts like the definition of Mengerian graphs [13, 11], and adaptations to temporal vertex disjoint versions [16, 10]. Because our problem is more closely related to edge connectivity, we refrain from commenting in detail the results on vertex connectivity, but refer the reader to [10] for an overview of such results. In what follows, we present the edge-related concepts and existing results. These are summarized in Table 1.

**Table 1** On the leftmost column, we specify the type of disjointness. Above, $\tau$ denotes the lifetime of the temporal graph, $k$ denotes the number of paths, $h$ denotes the size of the cut, NPc stands for NP-completeness, and W[1] or XP stands for W[1]-hardness or XP results when parameterizing by the size of the solution. Gray cells are proved in this paper.

| | Non-strict | | Strict | |
|---|---|---|---|---|
| | $\geq k$-Walks | $\leq h$-Cut | $\geq k$-Walks | $\leq h$-Cut |
| Multiedge | NPc [3], if $G$ dir., even for $k = \tau = 2$ | NPc for $\tau = 2$ (Theorem 13) | NPc for $\tau = 5$ [12] and $k = 2$ [15] | NPc for $\tau = 4$ [2] and W[1] for $h$ [8] |
| Edge | Polynomial [3] | | Polynomial [16] | |
| Snapshot | W[1] for $k$ (Th. 6) XP for $k$ (Th. 5) | W[1] for $h$ (Th. 7) XP for $h$ (Th. 4) | Open | |
| Node dep. | Open | | Polynomial [16] | |

A set of temporal $s, z$-walks are *edge disjoint* if they share no edges, and are *multiedge disjoint* if they share no multiedges, i.e, if no two walks in the set use consecutively a same pair of vertices. For example, in Figure 1a, the paths $(s, 1, w, 2, z)$ and $(s, 3, w, 3, u, 3, z)$ are edge disjoint, but are not multiedge disjoint, since they share the multiedge with endpoints $sw$. A set of (multi)edges is a *temporal (multi)edge $s, z$-cut* if they intersect every temporal $s, z$-walk. For example, in Figure 1a, $sw$ and $su$ form a multiedge $s, z$-cut, but if we want an edge $s, z$-cut, we have to pick both edges whose endpoints are $sw$. Some works define a temporal graph *with lifetime* $\tau$ as a pair $(G, \lambda)$ where $\lambda : E(G) \to 2^{[\tau]}$. Observe that there is equivalence with our model.

In [3], Berman showed that the edge problems for non-strict temporal paths are polynomial-time solvable, and that deciding the existence of at least $k$ multiedge disjoint temporal paths is NP-complete, $G$ directed or undirected, and if $G$ is directed, then the same holds even if $k = \tau = 2$. Up to our knowledge, no result concerning the cut problem related to multiedges is known. By a simple modification of a proof in [20], we present in Section 6 a proof of NP-completeness of the multiedge cut problem. Our proof also works for the case where $G$ is

a directed multigraph. Concerning strict paths, the complexities of these problems follow directly from results about problems on bounded length paths [2, 8]. Additionally, the strict problems related to edge disjoint paths was shown to be polynomial-time solvable in [16]. We mention that, among all the problems appearing in Table 1, Menger's Theorem holds only for edge disjoint paths in both the strict and non-strict contexts [3, 16], and node departure disjoint strict paths, defined below.

Another related concept is that of *node departure disjoint*, introduced in [16]. Given a temporal graph $(G, \lambda)$ with lifetime $\tau$, a set of strict temporal $s, z$-walks is *node departure disjoint* if no two of these paths leave a vertex in the same timestep. For example, in Figure 1a, $(s, 1, w, 2, z)$ and $(s, 3, w, 3, u, 3, z)$ are node departure disjoint. Additionally, a set $S \subseteq V(G) \times [\tau]$ is a *node departure $s, z$-cut* if each strict temporal $s, z$-walks contains an edge departing from $u$ in time $t$, for some $(u, t) \in S$. For example, in Figure 1a, the set $S = \{(s, 1), (s, 2), (s, 3)\}$ is a node departure $s, z$-cut. In [16], the authors prove that the maximum number of node departure disjoint $s, z$-walks is equal to the minimum size of a node departure $s, z$-cut. Even though the authors do not comment on the complexity of the related problems, their proof leads to a polynomial time algorithm as it consists of building a flow network and proving that the searched values are equivalent to applying the famous Maxflow-Mincut Theorem. Up to our knowledge, their results have not been investigated for the non-strict context.

The text is organized as follows. In Section 2, we present definitions, terminology and some basic results. In Section 3, we present our XP algorithms. In Section 4, we prove that $\leq h$-SNAPSHOT $s, z$-CUT and $\geq k$-SNAPSHOT DISJOINT TEMPORAL $s, z$-PATHS are W[1]-hard when parameterized by $h$ and $k$, respectively. In Section 5, we characterize Mengerian graphs. Finally, in Section 6, we prove that $\leq h$-Multiedge cut is NP-complete, and in Section 7 we present our concluding remarks.

## 2 Definitions and Terminology

Given positive integers $i, j \in \mathbb{N}$ such that $j \geq i$, we denote by $[i, j]$ the set $\{i, i+1, \ldots, j\}$ and by $[j]$ the set $\{1, \ldots, j\}$.

A *graph* is a triple $(V, E, f)$ where $V$ and $E$ are finite sets that we call *vertex set* and *edge set* respectively, and $f$ is a function that, for each $e \in E$ associates a pair $xy$ of elements in $V$, where $x \neq y$. We say that edge $e$ is *incident* to $x$ and $y$, that $x, y$ are the *endpoints* of $e$, and that $e$ *connects* $x$ and $y$. We omit $f$ in the rest of the paper and refer simply to the endpoints of $e$ instead. We also call the pair $xy$ a *multiedge*, and the number of edges with endpoints $xy$ is the *multiplicity* of the multiedge $xy$. If the multiplicity of each edge is 1, we say that $G$ is a *simple graph*. We denote by $U(G)$ the simple graph obtained from $G$ by decreasing the multiplicity of all multiedges to 1. See [18] for further basic definitions of graph theory.

Given a graph $G$ and a set of vertices $Z \subseteq V(G)$, the *identification of $Z$* is the graph obtained from $G - Z$ by adding a new vertex $z$ and, for every edge $e$ with endpoints $z'u$ where $z' \in Z$ and $u \notin Z$, add an edge $e'$ with endpoints $zu$. The graph $G'$ obtained from $G$ by a *subdivision* of an edge $e$ with endpoints $uv$ is the graph having $V(G) \cup \{z_e\}$ as vertex set, and $E(G - e) \cup \{e', e''\}$ as edge set, where $e'$ has endpoints $uz_e$ and $e''$ has endpoints $z_e v$. Finally, the graph obtained from $G$ by an *$m$-subidivision* of a multiedge $xy$ is the graph obtained by subdividing all the edges with endpoints $xy$ and then identifying the new vertices. Observe Figure 3 for an illustration of these definitions. The definition of m-subdivision has been introduced in [11]. Given a graph $H$, if $G$ has a subgraph that can be obtained from m-subdivisions of $H$, then we say that $H$ is an *$m$-topological minor* of $G$.

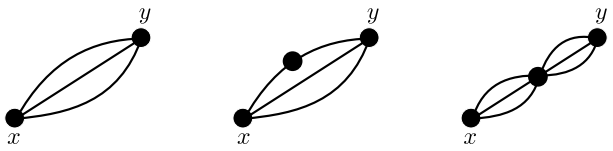



**Figure 3** From left to right: the multiedge $xy$, the subdivision of an edge with endpoints $xy$, and the m-subdivision of $xy$.

A *temporal graph* is a pair $(G, \lambda)$ where $G$ is a graph and $\lambda\colon E(G) \to \mathbb{N} \setminus \{0\}$. We refer to the elements of $\mathbb{N} \setminus \{0\}$ as *timesteps*. If an edge $e$ is such that $\lambda(e) = \alpha$ we say that $e$ is *active* or *appears* at timestep $\alpha$. A *temporal $x_1, x_q$-walk* in $(G, \lambda)$ is a sequence $P$ that alternates vertices and edges $(x_0, e_1, x_1, \ldots, e_q, x_q)$ such that for every $i \in [q]$, $e_i$ is an edge between $x_i$ and $x_{i-1}$ and $\lambda(e_1) \leq \cdots \leq \lambda(e_q)$. If $x_i \neq x_j$ for every $i, j \in [q]$ with $i \neq j$, we say that such temporal walk is a *temporal path*. Moreover, we define $V(P) = \{x_1, \ldots, x_q\}$ and $E(P) = \{e_1, \ldots, e_q\}$. For our purposes, we can assume that the subgraph active at a given timestep is simple, i.e., that if $e$ and $e'$ have both endpoints $xy$, then $\lambda(e) \neq \lambda(e')$. Such assumption allows us to define a path as a sequence of vertices and timesteps $(x_0, t_1, x_1, \ldots, t_{q-1}, x_q)$ such that, for each $i \in [q]$, there is an edge connecting $x_i x_{i-1}$ active at timestep $t_i$. The *lifetime* of $(G, \lambda)$ is denoted by $\tau(\lambda)$ and is the maximum integer such that there is an edge of $G$ active at such timestep. For each timestep $i \in \mathbb{N} \setminus \{0\}$, the *$i$-th snapshot* of $(G, \lambda)$ is the subgraph of $G$ defined as $H = (V(G), \lambda^{-1}(i))$.

Let $(G, \lambda)$ be a temporal graph with lifetime $\tau$. Also, let $s, z \in V(G)$ be vertices in $G$ and $Q, J$ temporal $s, z$-paths. We say that $Q$ and $J$ are *snapshot disjoint* if $\lambda(E(Q)) \cap \lambda(E(J)) = \emptyset$. A subset $S \subseteq [\tau]$ is a *snapshot $s, z$-cut* if every temporal $s, z$-path uses an edge active at some timestep in $S$. We denote by $sp_{G,\lambda}(s, z)$ the maximum number of snapshot disjoint temporal $s, z$-paths and by $sc_{G,\lambda}(s, z)$ the minimum size of a snapshot $s, z$-cut. Observe that if the above definitions are made in terms of temporal paths, then these parameters would not change. A graph $G$ is *Mengerian (for time)* if, for every timefunction $\lambda$ on $E(G)$, and every $s, z \in V(G)$, $s \neq z$, we have that $sp_{G,\lambda}(s, z) = sc_{G,\lambda}(s, z)$. The following will be useful later.

▶ **Proposition 2.** *If $G$ is non-Mengerian, then an m-subdivision of $G$ is also non-Mengerian.*

**Proof.** Let $G$ be a non-Mengerian graph and consider $\lambda \in E(G) \to \mathbb{N} \setminus \{0\}$ and $s, z \in V(G)$ to be such that $sp_{G,\lambda}(s, z) < sc_{G,\lambda}(s, z)$. Also, suppose that $H$ is obtained from $G$ by m-subdividing a multiedge, say $xy$. We construct a function $\lambda'$ from $\lambda$ that proves that $H$ is also non-Mengerian.

Let $D \subseteq E(G)$ be the set of edges of $G$ with endpoints $xy$, and denote by $v_{xy}$ the vertex of $H$ created by the m-subdivision of $xy$. Moreover, denote by $D_x$ and $D_y$ the sets of edges of $H$ with endpoints $xv_{xy}$ and $v_{xy}y$, respectively. Finally, define $\lambda'$ to be such that $\lambda'(e) = \lambda(e)$, for every $e \in E(G) \setminus D$, and $\lambda'(D_x) = \lambda'(D_y) = \lambda(D)$. We show that $sp_{G,\lambda}(s, z) = sp_{H,\lambda'}(s, z)$ and $sc_{G,\lambda}(s, z) = sc_{H,\lambda'}(s, z)$, which finishes our proof.

Given a set of snapshot disjoint temporal $s, t$-paths in $(G, \lambda)$, if some of these paths, say $P$, uses the edge $xy$, then in $(H, \lambda')$ we can substitute such edge by an edge in $D_x$ and another in $D_y$ active at the same time to obtain a temporal path $P'$ such that $V(P') = V(P) \cup \{v_{xy}\}$. This gives us a set of snapshot disjoint temporal $s, t$-paths in $(H, \lambda')$. In the other direction, if it is given a set of snapshot disjoint temporal $s, t$-paths in $(H, \lambda')$, if some of them uses the vertex $v_{xy}$, let $f_j$ be the edge used in $D_j$ for $j \in \{x, y\}$. Suppose without loss of generality that $\lambda(f_x) \geq \lambda(f_y)$. Then we substitute both edges incident to $v_{xy}$ by an edge in $D$ appearing at time $\lambda(f_x)$. This implies that $sp_{G,\lambda}(s, z) = sp_{G,\lambda'}(s, z)$. To see that $sc_{G,\lambda}(s, z) = sc_{H,\lambda'}(s, z)$ one can need to recall that $\lambda'(D_x) = \lambda'(D_y) = \lambda(D)$. ◀

▶ **Proposition 3.** *$G$ is Mengerian if and only if $H$ is Mengerian, for every subgraph $H$ of $G$.*

**Proof.** To prove necessity, suppose that $H \subseteq G$ is non-Mengerian, and let $s, z, \lambda$ be such that $sp_{H,\lambda}(s,z) < sc_{H,\lambda}(s,z)$. Consider the timefunction $\lambda'$ in $E(G)$ defined as follows.

$$\lambda'(e) = \begin{cases} \lambda(e) + 1 & \text{, for every } e \in E(H), \\ 1 & \text{, for every } e \in E(G) \setminus E(H) \text{ with endpoints } yt, \text{ and} \\ \max \lambda(E(H)) + 2 & \text{, otherwise.} \end{cases}$$

Because $H \subseteq G$ and $\lambda \subseteq \lambda'$, note that we get $sp_{H,\lambda}(s,z) \leq sp_{G,\lambda'}(s,z)$ and $sc_{H,\lambda}(s,z) \leq sc_{G,\lambda'}(s,z)$. Therefore it suffices to prove $sp_{G,\lambda'}(s,z) \leq sp_{H,\lambda}(s,z)$ and $sc_{G,\lambda'}(s,z) \leq sc_{H,\lambda}(s,z)$. Indeed, these hold because the timefunction $\lambda'$ does not allow for the existence of a temporal $s,t$-path not contained in $H$. ◀

## 3 Positive Results

In this section, we give XP algorithms for both SNAPSHOT $s,z$-CUT and SNAPSHOT DISJOINT TEMPORAL $s,z$-PATHS. Given the results of Section 4, unless FPT = W[1]-hard, XP algorithms are best possible from the point of view of parameterized complexity. The first algorithm is quite simple and consists of the usual approach of testing all possible cuts.

▶ **Theorem 4.** *Given a temporal graph $(G,\lambda)$ of lifetime $\tau$, a positive integer $h$ and $s, z \in V(G)$, we can solve $\leq h$-SNAPSHOT $s,z$-CUT in $O(\tau^h \cdot (|V(G)| + |E(G)|))$.*

**Proof.** Let $(G,\lambda), s, z, h$ as in the hypothesis of the theorem. For each subset $S \subseteq [\tau]$ of size $h$, define $G_S$ such that $V(G_S) = V(G)$ and $E(G_S) = \{e \in E(G) \mid \lambda(e) \notin S\}$. Define also $\lambda_S(e) = \lambda(e)$ for all $e \in E(G_S)$. Now, by the definition of $G_S$, any temporal $s,z$-path in $(G_S,\lambda_S)$ is a temporal $s,z$-path in $(G,\lambda)$ that does not use edges active at timesteps in $S$. Reciprocally, every temporal $s,z$-path in $G$ that does not use edges active at timesteps in $S$ is a temporal $s,z$-path in $(G_S,\lambda_S)$. As testing if there is a temporal $s,z$-path in $(G_S,\lambda_S)$ can be done in polynomial time [19] and $(G_S,\lambda_S)$ can be constructed in $O(n+m)$ time, it suffices to apply this test to $(G_S,\lambda_S)$ for every $S \subseteq [\tau]$ of size $h$. Since there are at most $\tau^h$ such sets, the theorem follows. ◀

The next algorithm is much more involved, and uses a technique similar to the one used to find disjoint paths between given pairs of vertices in a DAG [17].

▶ **Theorem 5.** *Given a temporal graph $(G,\lambda)$, vertices $s, z \in V(G)$ and a positive integer $k$, we can solve $\geq k$-SNAPSHOT DISJOINT TEMPORAL $s,z$-PATHS in time $O(m^k)$, where $m = |E(G)|$.*

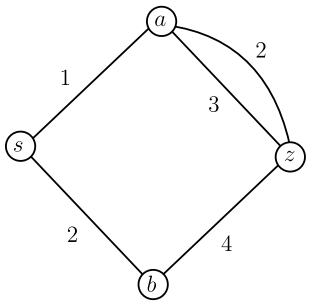
**Proof.** We construct a digraph $D$ with vertices $s^*$ and $z^*$ such that $|V(D)| = O(m^k)$ and there is an $s^*,z^*$-path in $D$ if and only if there are $k$ snapshot disjoint temporal $s,z$-paths in $(G,\lambda)$.

The vertex set of digraph $D$ is equal to the $k$-tuples formed by edges of $G$, together with vertices $s$ and $z$; formally $V(D) \subseteq F^k$, where $F = E(G) \cup \{s,z\}$. Vertex $s^*$ is set to be equal to $(s,\ldots,s)$, while vertex $z^*$ is set to be equal to $(z,\ldots,z)$. Each dimension of $V(D)$ represents one of the desired $k$ disjoint paths, and a set of snapshot disjoint temporal $s,z$-paths $P_1,\ldots,P_k$ will be represented by an $s^*,z^*$-path $P$ in $G$, as previously said. So $s^*$ represents the starting point, and $z^*$ represents the finish point of every temporal $s,z$-path. Then, when an edge of $D$ is traversed by $P$, we want that at least one of the $k$ paths traverses an edge. Because we want to allow that only one of the paths gets closer to $z$ with each step

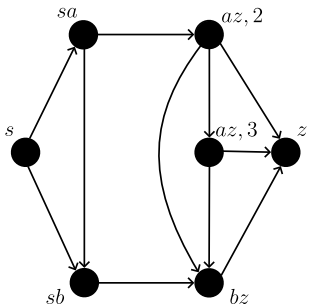
of $P$, there will be an edge from $\alpha \in V(D)$ to $\beta \in V(D)$ only if exactly one position of $\alpha$ and $\beta$ differ. Not only this, but we want that, at each step of $P$, the path $P_i$ that gets closer to $z$ is the one whose last traversed edge occurs the earliest among all the $P_i$'s. In the next paragraph, we formally construct digraph $D$.

As previously said, let $F = E(G) \cup \{s, z\}$. Throughout the construction, we will be referring to Figures 4 and 5. Because we want to avoid simultaneous traversal of paths that intersect in a snapshot, we only consider elements of $F^k$ whose each pair of coordinates are active in different snapshots. Indeed, if $k = 2$ and we allow for instance the existence of vertex $(e, e')$ such that $t = \lambda(e) = \lambda(e')$, then this would mean that the constructed paths $P_1$ and $P_2$ intersect in timestep $t$. Therefore, we define $V$ as formalized below. Observe that this implies, in Figure 5, that vertices $\{(e, e) \mid e \in E(G)\} \cup \{(az_2, sb), (sb, az_2)\}$ do not exist in $V(D)$, where $az_2$ denotes the edge with endpoints $az$ active in timestep 2.

$$V = \{(u_1, \ldots, u_k) \in F^k \mid \forall i, j \in [k] \text{ with } i \neq j, \text{ we have } \lambda(u_i) \neq \lambda(u_j) \text{ or } u_i = u_j \in \{s, z\}\}$$




**(a)** Example of temporal graph $(G, \lambda)$.




**(b)** Auxiliary graph $M$ on the set $F = E(G) \cup \{s, z\}$.

**Figure 4** Example of construction in Theorem 5.

Now, we define the edge set of $D$. For this, we first construct an auxiliary graph $M$ whose vertex set is equal to $F$; observe Figure 4b to follow the construction. First of all, we want that a traversal of an edge in $D$ translates into a valid traversal in $(G, \lambda)$. Therefore, for every pair $e, f \in F$, add to $D$ an edge from $e$ to $f$ only if $e$ can be followed by $f$ in a temporal $s, z$-path in $(G, \lambda)$. Formally, add $ef$ in the following cases:

- For every $e \in E(G)$, and every $f \in E(G)$ adjacent to $e$ such that $\lambda(e) \leq \lambda(f)$;
- For $e = s$ and every $f \in E(G)$ incident to $s$; and
- For every $e \in E(G)$ incident to $z$ and $f = z$.

Finally, as previously said, we want that at each edge traversal of an $s^*, z^*$-path in $D$, the path in $(G, \lambda)$ that is getting closer to $z$ is that one whose last used edge is the earliest (one with the smallest value of $\lambda$) among all the other paths. To help with this, we also define $\lambda(s)$ to be equal to 0, and $\lambda(z)$ to be equal to $\tau + 1$, where $\tau$ is equal to the lifetime of $(G, \lambda)$. This means intuitively that we give always priority to leave $s$, and that, once we reach $z$ in any dimension, then we cannot depart from $z$ anymore. So, given a vertex $\alpha = (u_1, \ldots, u_k) \in V(D)$, we add an edge from $\alpha$ to $\beta \in V(D)$ if and only if:

- $\beta$ differ from $\alpha$ in exactly one position, $i$;
- $i$ is such that $\lambda(u_i) \leq \min_{j \in [k]} \lambda(u_j)$; and
- By letting $u'_i$ be the value in the $i$-th position of $\beta$, we have that $u_i u'_i$ is a valid move, i.e., that $u_i u'_i \in E(M)$.

Observe Figure 5. Another way of seeing this construction is by starting with copies of $M$ on each row and column of $D$, then removing the vertices that do not belong to $D$, and finally removing from row/column $e$ any edge leaving $f$ with $\lambda(f) > \lambda(e)$.

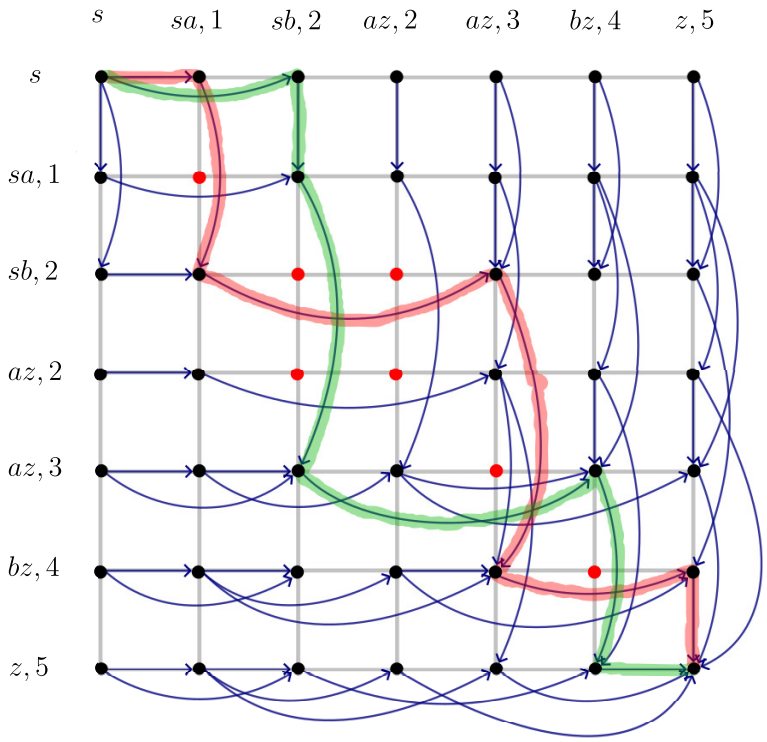



**Figure 5** Digraph $D$ related to the temporal graph in Figure 4a; value $k = 2$ is being used, which means that $V(D) \subseteq F^2$. Each row and column is labeled with an element $e$ of $F$, together with the value $\lambda(e)$; this will help in the construction. A vertex $(e, f)$ of $D$ is represented in the intersection of row $e$ and column $f$. Red dots in the figure represent the fact that the related pair $(row, column)$ is not a vertex in $D$. The snapshot disjoint temporal $s, z$-paths $P_1 = (s, 1, a, 3, z)$ and $P_2 = (s, 2, b, 4, z)$ can be obtained either through the red or the green $s^*, z^*$-path.

Now, we prove that there are $k$ snapshot disjoint temporal $s, z$-paths in $(G, \lambda)$ if and only if there is an $s^*, z^*$-path in $D$. In what follows, given a vertex $\alpha \in V(D)$, we denote by $(a_1, \ldots, a_k)$ the tuple related to $\alpha$. Recall that $s^* = (s, \ldots, s)$ and $z^* = (z, \ldots, z)$. Suppose $P_1, \ldots, P_k$ is a set of snapshot disjoint temporal $s, z$-paths in $(G, \lambda)$. For each $i \in [k]$, let $e_i^1, \ldots, e_i^{p_i}$ be the sequence of edges used in $P_i$ in order of traversal and define $e_i^0 = s$ and $e_i^{p_i+1} = z$. By induction, we define a sequence of vertices of $D$, $(s^* = \alpha^1, \ldots, \alpha^q = \alpha)$, that forms an $s^*, \alpha$-path for some $\alpha$ with the following property:

**(P)** For each dimension $i \in [k]$ and each $\ell \in [q]$, the sequence of edges traversed in dimension $i$ is a subpath of $P_i$. Formally, by removing $s$ and repetitions of edges from the sequence $(a_i^1, \ldots, a_i^q)$, we obtain a subsequence of $e_i^1, \ldots, e_i^{p_i}$.

First, we define $\alpha_1 = s^*$; clearly property (P) holds as all paths start in $s$. Now suppose that sequence $\alpha_1, \ldots, \alpha_q$ satisfying Property (P) is obtained, $q \geq 1$. Let $i \in [k]$ be such that $\lambda(a_i^q) = \min_{j \in [k]} \lambda(a_j^q)$. By Property (P), observe that either $a_i^q = s$, or $a_i^q$ is an edge of $P$, or $a_i^q = z$. If the latter occurs, then we have that $P$ is an $s^*, z^*$-path in $D$, since $\lambda(z) > \lambda(e)$ for every $e \in F \setminus \{z\}$, i.e., the only way $\lambda(z)$ is minimum is if all other positions are also equal to $z$. So suppose one of the other cases occurs. Note that it means that there is some edge

following $a_i^q$ in $P$, say $e_i^\ell$. By definition of temporal path, $\lambda(e_i^\ell) \geq \lambda(a_i^q)$; hence $a_i^q e_i^\ell \in E(M)$. Define $\alpha^{q+1}$ to be equal to $\alpha^q$ except that in position $i$ we have $e_i^\ell$ instead of $a_i^q$, and note that $(\alpha^1, \ldots, \alpha^{q+1})$ is a path in $D$ that satisfies Property (P).

Now suppose the existence of an $s^*, z^*$-path in $D$, $(\alpha^1, \ldots, \alpha^q)$. We construct a set of $k$ snapshot disjoint temporal $s, z$-paths in $(G, \lambda)$. For this, for each $i \in [k]$, let $P_i$ be a sequence of edges obtained from dimension $i$, i.e., from $(a_i^1, \ldots, a_i^q)$ by removing occurrences of $s$ and $z$, and repetitions of edges. Because each transition respects $M$, we trivially get that $P_i$ defines a temporal $s, z$-path in $(G, \lambda)$. It remains to show that such paths are snapshot disjoint. Suppose otherwise, and let $i, j$ be such that there are edges $e_i$ in $P_i$ and $e_j$ in $P_j$ such that $\lambda(e_i) = \lambda(e_j) = \ell$. Let $\ell_i$ be the smallest index such that $a_i^{\ell_i} = e_i$, and $\ell_j$ be the smallest index such that $a_j^{\ell_j} = e_j$. By the definition of $V(D)$, we have that $\ell_i \neq \ell_j$. Indeed no vertex of $D$ can contain two elements of $F$ with same value of $\lambda$, and recall that $i \neq j$ as $P_i, P_j$ are distinct paths. So, we can suppose, without loss of generality, that $\ell_i < \ell_j$. Observe that this means that $\alpha^{\ell_i - 1}$ differ from $\alpha^{\ell_i}$ in exactly position $i$; additionally, it means that $\lambda(e_i) \leq \min_{h \in [k]} \lambda(a_h^{\ell_i - 1})$. In particular, we have that $\ell = \lambda(e_i) \leq \lambda(a_j^{\ell_i - 1})$. But observe that, in a fixed dimension, the values of $\lambda$ can only increase, i.e., since $\ell_i < \ell_j$, we get $\lambda(a_j^{\ell_i - 1}) \leq \lambda(a_j^{\ell_j}) = \lambda(e_j) = \ell$. We get a contradiction as in this case vertex $\alpha^{\ell_i - 1}$ should not be defined as it contains two elements with the same value of $\lambda$, namely $a_j^{\ell_i - 1}$ and $e_i$.

To finish the proof just recall that $|V(D)| \leq (m+2)^k$, where $m = |E(G)|$, and that deciding if there is a path between two vertices in $D$ can be made in time $O(|V(D)|^2)$. So, deciding if there are $k$ snapshot disjoint temporal $s, z$-paths in $(G, \lambda)$ can be done in time $O(m^k)$. ◀

## 4 Negative Results

In this section, we prove that the algorithms presented in Section 3 are best possible, i.e., that $\geq k$-SNAPSHOT DISJOINT TEMPORAL $s, z$-PATHS and $\leq h$-SNAPSHOT $s, z$-CUT are W[1]-hard when parameterized by $k$ and $h$, respectively.

▶ **Theorem 6.** *$\geq k$-SNAPSHOT DISJOINT TEMPORAL $s, z$-PATHS is W[1]-hard when parameterized by $k$.*

**Proof.** We make a parameterized reduction from $\geq k$-INDEPENDENT SET when parameterized by $k$. Such problem has as input a simple graph $G$ and an integer $k$, and the question is whether $G$ has an independent set of size at least $k$. This is known to be W[1]-hard (see e.g. [6]).

Consider an instance $G, k$ of $\geq k$-INDEPENDENT SET and let $|V(G)| = n$. Observe Figure 6 to follow the construction. First, add to $G'$ vertices $s$ and $z$. Then, for each $u \in V(G)$, add to $G'$ an $s, z$-path on $d(u)$ edges; denote such path by $Q_u$. Now, consider any ordering $e_1, \ldots, e_m$ of $E(G)$, and denote the edges incident to a vertex $u \in V(G)$ by $\delta(u)$. We can define $\lambda\colon E(G') \to \mathbb{N} \setminus \{0\}$ in a way that each $Q_u$ is a temporal $s, z$-path using the orders of the edges in $\delta(u)$. Formally, for each $u \in V(G)$, let $\delta(u) = \{e_{i_1}, \ldots, e_{i_q}\}$ with $i_1 < \ldots < i_q$, and define $\lambda(E(Q_u))$ to be equal to $\{i_1, \ldots, i_q\}$ in a way that $Q_u$ is a temporal path.

We show that $(G', \lambda)$ has $k$ snapshot disjoint temporal $s, z$-paths if and only if $G$ has an independent set of size at least $k$. By the definition of $G'$, all temporal $s, z$-paths are of type $Q_u$ for some $u \in V(G)$. Therefore, it suffices to show that a subset $S \subseteq V(G)$ is an independent set of $G$ if and only if $\{Q_u \mid u \in S\}$ is a set of snapshot disjoint temporal $s, z$-paths in $(G', \lambda)$. Suppose first that $S$ is an independent set, and suppose by contradiction that $u_1, u_2 \in S$ are such that $Q_{u_1}$ and $Q_{u_2}$ are not snapshot disjoint. Then there exists

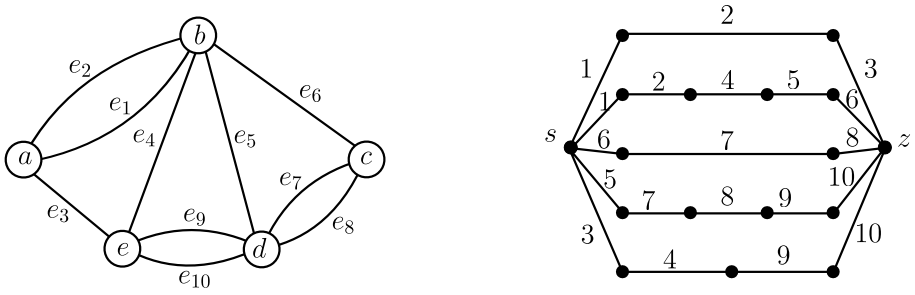



**Figure 6** To the left, graph $G$, and to the right, the constructed temporal graph $(G', \lambda)$. In $G'$, paths $P_a, P_b, P_c, P_d, P_e$ are depicted from top to bottom, in this order.

$e_1 \in E(Q_1)$ and $e_2 \in E(Q_2)$ such that $\lambda(e_1) = \lambda(e_2)$. By construction, this means that $e_1 = e_2$, and since $u_1 \neq u_2$, we get that actually this edge has endpoints $u_1u_2$, a contradiction as $S$ is an independent set. Thus, $\{Q_u \mid u \in S\}$ is a set of snapshot disjoint temporal $s, z$-paths. Finally, observe that if $e_i = vw \in E(G)$, then $\lambda(Q_v) \cap \lambda(Q_w) = \{i\}$, which directly implies that if $\{Q_u \mid u \in S\}$ is a set of snapshot disjoint temporal $s, z$-paths, then $S$ cannot contain any pair of adjacent vertices. ◀

Now, we prove the analogous result for the cut problem.

▶ **Theorem 7.** $\leq h$-SNAPSHOT $s, z$-CUT *is* W[1]*-hard when parameterized by* $h$.

**Proof.** We make a reduction from MULTICOLORED $k$-CLIQUE, when parameterized by $k$, known to be W[1]-hard [6]. Such problem has as input a simple graph $G$, an integer $k$, and a partition of $V(G)$ into $k$ independent sets (alternatively, a proper $k$-coloring), and the question is whether $G$ has a (multicolored) clique of size $k$. So let $G$ be a graph and $\{X_1, \ldots, X_k\}$ be a proper $k$-coloring of $G$. By adding artificial vertices and edges if necessary, we can suppose that the number of edges between $X_i$ and $X_j$ is equal to a value $m$, for every pair $i, j \in [k]$. So, for $i, j \in [k]$, $i \neq j$, denote the set of such edges by $E_{i,j} = \{e^{i,j}_1, \ldots, e^{i,j}_m\}$. We make this assumption in order to make presentation simpler.

Now, for each $i, j \in [k]$, $i \neq j$, we construct a gadget denoted by $F_{i,j}$. Observe Figure 7 to follow the construction. First add to $F_{i,j}$ the set of vertices $V_{i,j} = \{v^{i,j}_0, \ldots, v^{i,j}_{2m}\}$, making the first $m + 1$ of them form a path of multiplicity $m$, and the latter $m + 1$ form a path of multiplicity 1. Formally, for each $\ell \in \{0, \ldots, m - 1\}$, add $m$ edges with endpoints $v^{i,j}_\ell v^{i,j}_{\ell+1}$. Also, for each $\ell \in \{m, \ldots, 2m-1\}$, add 1 edge with endpoints $v^{i,j}_\ell v^{i,j}_{\ell+1}$. Now, for each $\ell \in [m]$, add vertex $w^{i,j}_\ell$ and join such vertex with $v^{i,j}_\ell$ by a path with $m - 1$ edges and denote such path by $P^{i,j}_\ell$. We say that vertex $w^{i,j}_\ell$ of our gadget is *associated with edge* $e^{i,j}_\ell$ of $E_{i,j}$. The timefunction is defined only later.

Now, we finish the construction of our temporal graph. For this, take the union of all graphs $F_{i,j}$ and identify all vertices $v^{i,j}_0$, calling the obtained vertex $s$, and identify all vertices $v^{i,j}_{2m}$, calling the obtained vertex $z$. Also, for each $i, j \in [k]$, $i \neq j$, and $\ell \in [m]$, we add two edges between $w^{i,j}_\ell$ and $z$. Denote by $G'$ the obtained graph, and by $W$ the set $\{w^{i,j}_\ell \mid i, j \in [k], i \neq j, \ell \in [m]\}$. Observe that $G'$ contains $O(k^2 \cdot m)$ vertices and edges.

Now we define $\lambda$. The idea is that each $F_{i,j}$ will be active during its own dedicated time window. Formally, define $\Delta_{i,j} = [f_{i,j} + 1, f_{i,j} + m]$, for each pair $i, j \in [k]$, $i \neq j$, in a way that $\Delta_{i,j} \cap \Delta_{i',j'} = \emptyset$ whenever $\{i, j\} \neq \{i', j'\}$. Now, consider $i, j \in [k]$ with $i \neq j$. For each $\ell \in \{0, \ldots, m - 1\}$, we define $\lambda$ in a way that every value in $\Delta_{i,j}$ appears in some edge with endpoints $v^{i,j}_\ell v^{i,j}_{\ell+1}$. Also, for each $\ell \in [m]$, we let $\lambda(v^{i,j}_{m-1+\ell} v^{i,j}_{m+\ell}) = \{f_i + \ell\}$, and we define

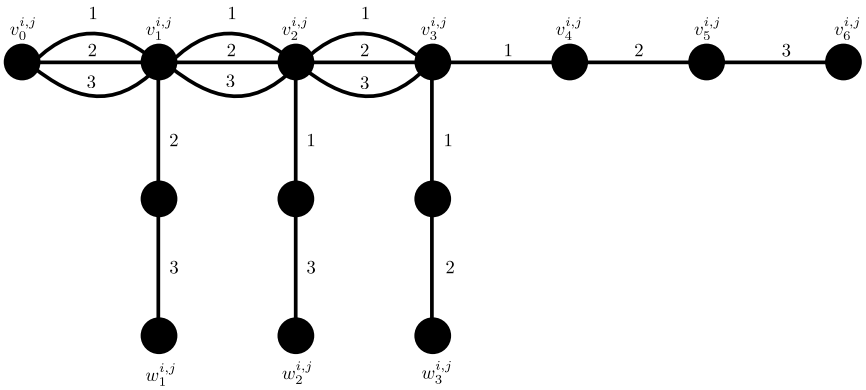



**Figure 7** A representation of $F_{i,j}$ with labels of $\lambda$ where $m = 3$ and $\Delta_{i,j} = \{1, 2, 3\}$.

$\lambda(E(P_\ell^{i,j}))$ to be equal to $\Delta_{i,j} \setminus \{f_{i,j} + \ell\}$ and in a way that $P_\ell^{i,j}$ is a temporal $v_\ell^{i,j}, w_\ell^{i,j}$-path. Finally, the only edges that remain unlabelled are the edges between $z$ and vertices of type $w$. For such edges, we reserve a time window of size $n = |V(G)|$, that we denote by $\Delta_V$, where any timestep in such set is greater that any timestep we used to define $\lambda$ so far. Moreover, we associate each vertex $v \in V(G)$ with a timestep $t_v \in \Delta_V$. Let $w_\ell^{i,j} \in W$ and recall that such vertex is associated with $e_\ell^{i,j} \in E(G)$. Suppose $e_\ell^{i,j}$ have endpoints $xy$, and let the two edges of $G'$ with endpoints $w_\ell^{i,j}z$ be active in timesteps $\{t_x, t_y\}$.

Now, we prove that $G$ has a clique of size $k$ if and only if $(G', \lambda)$ has a snapshot $s, z$-cut of size at most $\binom{k}{2} + k$. Consider first a clique $C$ of $G$ of size $k$, and let $\{e_{\ell_1}^{i_1,j_1}, \ldots, e_{\ell_a}^{i_a,j_a}\}$ be the set of edges of $G$ between vertices of $C$. Notice that, because $C$ has a vertex from each part, we get that $a = \binom{k}{2}$. Define $S = \{f_{i_b,j_b} + \ell_b \mid b \in \{1, \ldots, a\}\} \cup \{t_v \mid v \in C\}$. We prove that $S$ is a snapshot $s, z$-cut. By contradiction, suppose that $P$ is a temporal $s, z$-path not passing by $S$, i.e., such that $\lambda(E(P)) \cap S = \emptyset$. Since $a = \binom{k}{2}$ and all edges incident to $s$ are active in timesteps $\bigcup_{i,j \in [k], i \neq j} \Delta_{i,j}$, we can define $b \in [a]$ to be the index related to the first edge in $P$, i.e., $P$ starts in an edge of $F_{i_b,j_b}$, say the one active in timestep $f_{i_b,j_b} + \ell_b$. Observe that the value $f_{i_b,j_b} + \ell_b$ is within the temporal $s, z$-path contained in $F_{i_b,j_b}$, and that it also separates $s$ and $w_\ell^{i_b,j_b}$ for every $\ell \in [m] \setminus \{\ell_b\}$. Hence, $P$ must start with the temporal $s, w_{\ell_b}^{i_b,j_b}$-path contained in $F_{i_b,j_b}$. However, as $e_{\ell_b}^{i_b,j_b}$ is incident to vertices of the clique, say $x$ and $y$, then we have that $P$ uses timestep $t_x$ or $t_y$, a contradiction as $\{t_x, t_y\} \subseteq S$.

Now, suppose that $S$ is a minimum snapshot $s, z$-cut in $(G', \lambda)$ and that it has size at most $\binom{k}{2} + k$. Let $V_S = \{x \in V(G) \mid t_x \in S\}$. We prove that $V_S$ is a clique of $G$ of size $k$. Denote by $O$ the set of pairs $\{(i, j) \mid i, j \in [k], i < j\}$. We say that $(i, j) \in O$ is *open* if $\Delta_{i,j} \cap S = \{f_{i,j} + \ell\}$ for some $\ell \in [m]$, and we say that $e_\ell^{i,j}$ is the *open edge of* $(i, j)$. The following simple facts will be useful:

1. For every $i, j \in [k]$, $i \neq j$, we have $\Delta_{i,j} \cap S \neq \emptyset$: this is due to the fact that there is a temporal $s, z$-path using only timesteps in $\Delta_{i,j}$;
2. If $\ell \in [m]$ is such that $\Delta_{i,j} \cap S = \{f_{i,j} + \ell\}$, then $\{x, y\} \subseteq V_S$, where $xy$ are the endpoints of $e_\ell^{i,j}$: this is because there exists a temporal $s, w_\ell^{i,j}$-path not using any timestep in $S$, and hence such path can be extended to a temporal $s, z$-path by using an edge with endpoints $w_\ell^{i,j}$ either in timestep $t_x$ or in timestep $t_y$;
3. For every $i, j \in [k]$, $i \neq j$, we have $|\Delta_{i,j} \cap S| \leq 2$: it suffices to see that any two timesteps in $\Delta_{i,j}$ intersects all temporal paths between $s$ and any vertex in $\{w_\ell^{i,j} \mid \ell \in [m]\} \cup \{z\}$;
4. If $x \in V_S$, then $x$ is incident to some open edge: indeed, if $x$ is not incident to any open edge, then $w_\ell^{i,j}$ is separated from $s$ by $S \setminus \{t_x\}$ for every edge $e_\ell^{i,j}$ incident in $x$, and since timestep $t_x$ contains only edges incident to some such $w_\ell^{i,j}$, it follows that $S \setminus \{t_x\}$ is also a snapshot $s, z$-cut, contradicting the minimality of $S$.

By Fact 3, if $(i,j)$ is not open, then $\Delta_{i,j} \cap S = \{f_{i,j}+\ell_1, f_{i,j}+\ell_2\}$ for some pair of values $\ell_1, \ell_2 \in [m]$. In such case, we say that edges $e^{i,j}_{\ell_1}$ and $e^{i,j}_{\ell_2}$ are *chosen* for $(i,j)$. We show how to modify $S$ in order to decrease the number of chosen edges.

▷ Claim 8. If $(i,j)$ is not open, then we can suppose that $V_S \cap (X_i \cup X_j) = \emptyset$.

Proof of Claim 8. Let $e^{i,j}_{\ell_1}$ and $e^{i,j}_{\ell_2}$ be the chosen edges for $(i,j)$. Denote by $x_1y_1$ and $x_2y_2$ the endpoints of $e^{i,j}_{\ell_1}$ and $e^{i,j}_{\ell_2}$, respectively. Suppose without loss of generality that $\{x_1,x_2\} \subseteq X_i$ and $\{y_1,y_2\} \subseteq X_j$. Now suppose that there exists $x \in X_i \cap S$. If $x \in \{x_1,x_2\}$, say $x = x_1$, then replace $f_{i,j}+\ell_2$ in $S$ by $y_1$, obtaining $S'$. Observe that in this case $e^{i,j}_{\ell_1}$ becomes an open edge, but such that $\{t_{x_1}, t_{y_1}\} \subseteq S'$. Hence $S'$ is still a snapshot $s,z$-cut containing fewer chosen edges. And if $x \notin \{x_1,x_2\}$, then let $xy$ be any edge incident in $x$ such that $y \in X_j$. We can suppose that such edge exists as otherwise $x$ cannot be in any multicolored $k$-clique and then we can remove it from $G$. Let $\ell \in [m]$ be such that $e^{i,j}_{\ell}$ has endpoints $xy$. Observe that Fact 3 also tells us that $S' = (S \setminus \{f_{i,j}+\ell_2\}) \cup \{f_{i,j}+\ell\}$ is a snapshot $s,z$-cut. We can then apply the previous argument to replace $f_{i,j}+\ell_1$ by $t_y$ to again obtain a snapshot $s,z$-cut with fewer chosen edges. ◁

▷ Claim 9. We can suppose that every pair is open.

Proof of Claim 9. We can clearly suppose that $G$ is connected, as otherwise the answer to Multicolored $k$-Clique is trivially "no". Now, let $I_1$ be the set of indices $\{i \in [k] \mid V_S \cap X_i = \emptyset\}$ and $I_2 = S \setminus I_1$. Suppose first that $I_1 \neq \emptyset$ and $I_2 \neq \emptyset$, and consider $i_1 \in I_1$ and $i_2 \in I_2$. By Claim 8 and because $i_2 \in I_2$, we know that $(i_1,i_2)$ is open. So let $e^{i_1,i_2}_{\ell}$ be the open edge for $(i_1,i_2)$, and let $xy$ be its endpoints, with $x \in X_{i_1}$ and $y \in X_{i_2}$. By Fact 2, we get that $\{t_x,t_y\} \subseteq S$, i.e., $\{x,y\} \subseteq V_S$, contradicting Claim 8 as $x \in X_{i_1}$ and $i_1 \in I_1$.

So either $I_1 = \emptyset$ or $I_2 = \emptyset$. Observe that if $I_1 = \emptyset$, then $V_S \cap X_i \neq \emptyset$, for every $i \in [k]$, and the claim follows from Claim 8. And if $I_2 = \emptyset$, then by Fact 3 we get that $S$ contains two edges for every pair $i,j$, totalling $|S| = 2\binom{k}{2}$. Since $|S| \leq \binom{k}{2} + 2$, we get that this happens only if $k \leq 3$, in which case Multicolored $k$-Clique is polynomial-time solvable. ◁

Finally, observe that the set of open edges, $E^*$, contains exactly $\binom{k}{2}$ edges, by definition of open edge and by Claim 9. We then get that $|S| = |E^*| + |V_S| = \binom{k}{2} + |V_S|$. It follows that $|V_S| \leq k$. Additionally, by Fact 2 we know that $E^*$ forms a subgraph of $G$ with vertex set $V_S$. Because $G$ is a simple graph, $E^*$ contains $\binom{k}{2}$ edges, and $V_S$ contains at most $k$ vertices, the only way this can be possible is if $V_S$ contains exactly $k$ pairwise adjacent vertices, i.e., $V_S$ is a clique of size $k$, as we wanted to prove. ◀

## 5 Characterization and recognition of Mengerian graphs

For a temporal graph $(G,\lambda)$ and pair of vertices $s,z$, let $\mathcal{P}$ be a set of snapshot disjoint temporal $s,z$-paths and $S$ a snapshot $s,z$-cut. By definition, for each path $P \in \mathcal{P}$, there is an edge in $P$ active at a timestep $t$, for some $t \in S$; choose $\alpha(P)$ to be one such timestep. As the paths in $\mathcal{P}$ are snapshot disjoint , we have that $\alpha(P) \neq \alpha(Q)$ for every $Q \in \mathcal{P}$ different from $P$. Therefore $|\mathcal{P}| \leq |S|$ and the inequality $sp_{G,\lambda}(s,z) \leq sc_{G,\lambda}(s,z)$ follows. In Proposition 10 we show that there are temporal graphs for which $sp_{G,\lambda}(s,z) < sc_{G,\lambda}(s,z)$, and after this we prove Theorem 1. In our characterization, we have 5 graphs as forbidden structures, $M_1, M_2, M_3, M_4, M_5$, that are represented in Figure 2. Let $\mathcal{M}$ be the set of such graphs.

In Figure 8, we present timefunctions for the graphs in $\mathcal{M}$ that turn the inequality $sp_{G,\lambda}(s,z) \leq sc_{G,\lambda}(s,z)$ strict. This is formally stated in the next proposition.

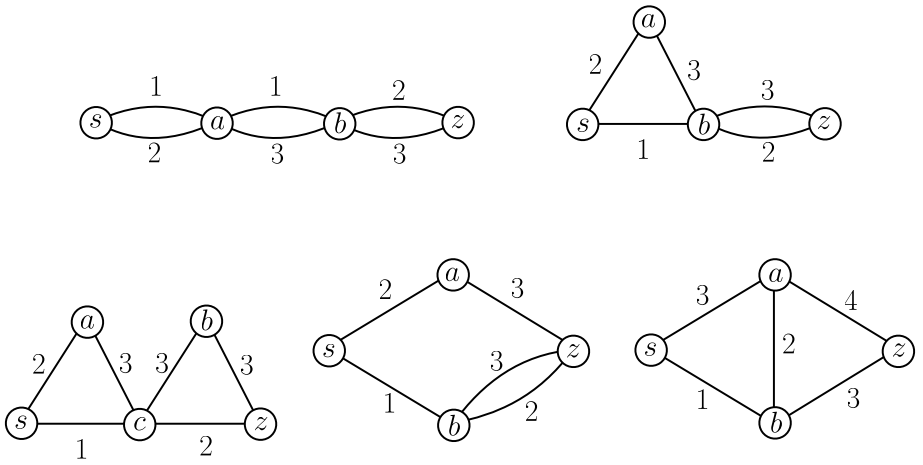



**Figure 8** Graphs in the set $\mathcal{M}$ with timefunctions such that $sp_{G\lambda}(s,z) < sc_{G,\lambda}(s,z)$.

▶ **Proposition 10.** *Let $(G,\lambda)$ be one of the temporal graphs depicted in Figure 8. Then $sp_{G,\lambda}(s,z) < sc_{G,\lambda}(s,z)$.*

**Proof.** To observe that $sc_{G,\lambda}(s,z) > 1$, one just needs do verify that for each timestep, there is a temporal $s,z$-path not using this timestep. Now, for the cases $G \in \{M_1, M_2, M_3, M_4\}$, suppose $sp_{G,\lambda}(s,z) = sc_{G,\lambda}(s,z)$ and let $Q, J$ be snapshot disjoint temporal $s,z$-paths. In each of those cases, there are only two edges incident to $s$, one active at timestep 1 and other at timestep 2. Hence, one of these paths, say $Q$, starts at timestep 2. Observe that in this case, $Q$ can only finish through edges active at timestep 3. Therefore, $J$ cannot use timestep 2 nor 3, however, all edges incident to $z$ are active at timesteps 2 or 3, a contradiction as there is no temporal $s,z$-path contained in the first snapshot. Finally, suppose $G = M_5$; we will apply a similar argument. So, let $Q, J$ be snapshot disjoint temporal $s,z$-paths. Note that one of them, say $Q$, must use the edge incident to $s$ active at timestep 3 and, therefore, finishes using the edge incident to $z$ active at timestep 4. It follows that $J$ is not allowed to use timestep 3 nor 4, a contradiction as all edges incident to $z$ are only active at such timesteps. ◀

We now prove that the equality between the parameters always holds if each timestep contains at most one active edge, i.e., $\lambda$ is injective.

▶ **Proposition 11.** *Let $(G,\lambda)$ be a temporal graph such that $\lambda$ is injective. Then $sp_{G,\lambda}(s,z) = sc_{G,\lambda}(s,z)$ for every $s,z \in V(G)$. Moreover, we can compute such value in polynomial time.*

**Proof.** If $P$ and $Q$ are snapshot disjoint temporal $s,z$-paths, then, for $e \in E(P)$ and $f \in E(Q)$, we have that $\lambda(e) \neq \lambda(f)$, therefore $e \neq f$. On other hand, if $P$ and $Q$ are such that $E(P) \cap E(Q) = \emptyset$, then we have that $\lambda(E(P)) \cap \lambda(E(Q)) = \emptyset$. Thus, $P$ and $Q$ are snapshot disjoint if and only if they are edge disjoint. Therefore, the maximum size of edge disjoint temporal $s,z$-paths is equal $sp_{G,\lambda}(s,z)$. Moreover notice that $S \subseteq E(G)$ is a set such that every temporal $s,z$-path uses an edge of $S$, then every temporal $s,z$-path uses an edge active at timestep $\{\lambda(e) \mid e \in S\}$. If $S^T \subseteq [\tau]$ is such that every temporal $s,z$-path uses an edge active at timestep $\alpha \in S^T$, then it uses the only edge in $\lambda^{-1}(\alpha)$. Therefore, the size of a minimum set of edges such that every temporal $s,z$-path intersects such set is equal to $sc_{G,\lambda}(s,z)$. Using a result proved in [3] we conclude that $sp_{G,\lambda}(s,z) = sc_{G,\lambda}(s,z)$, and that both parameters can be found in polynomial time. ◀

**Proof of Theorem 1.** We first prove that if $G$ has $M$ as a m-topological minor, for some $M \in \mathcal{M}$, then $G$ is not Mengerian. In Proposition 10, we already get that each $M \in \mathcal{M}$ is not Mengerian. To finish necessity, we need to prove that adding vertices, adding edges and m-subdividing edges in $H$ does not lead to a Mengerian graph. Propositions 2 and 3 do this work.

Now, suppose by contradiction that $G$ does not contain any $M$ in $\mathcal{M}$ as m-topological minor and that $G$ is non-Mengerian. Hence, there exists $\lambda\colon E(G) \to \mathbb{N} \setminus \{0\}$ and $s, z \in V(G)$ such that $sp_{G,\lambda}(s,z) < sc_{G,\lambda}(s,z)$. Suppose, without loss of generality, that among all such graphs and timefunctions we choose those that minimize $|V(G)| + |E(G)| + \tau(\lambda)$. We first show that there are no edges connecting $s$ and $z$. Suppose otherwise, that there is an edge $e$ with endpoints and let $\lambda(e) = \alpha$. Let $G'$ be the graph obtained from removing all edges in $\lambda^{-1}(\alpha)$ and $\lambda'$, the restriction of $\lambda$ to $E(G) \setminus \lambda^{-1}(\alpha)$. Because $(s, e, z)$ is a temporal $s, z$-path, observe that $sp_{G',\lambda'}(s,z) = sp_{G,\lambda}(s,z) - 1$ and $sc_{G',\lambda'}(s,z) = sc_{G,\lambda}(s,z) - 1$. This contradicts the fact that $(G, \lambda)$ minimizes $|V(G)| + |E(G)| + \tau(\lambda)$. Now, suppose that there is a cycle $C$ containing $s$ and $z$. As $sz \notin E(G)$, we have that such cycle has size at least 4. As $G$ has no $M_5$ as m-topological minor, there are no paths between the vertices of $C$ that are not contained in $C$. Also, as $G$ has no $M_4$ as m-topological minor, all the multiedges of $C$ have multiplicity 1. In other words, the 2-connected component containing $C$ is formed just by $C$ itself, with each multiedge of C having multiplicity 1. In particular, there are only two paths connecting $s$ and $z$. If these two paths are snapshot disjoint temporal $s, z$-paths, we take the timesteps of the two edges incident to $z$ to obtain a snapshot $s, z$-cut as it kills the only two temporal $s, z$-paths. This is a contradiction as we then get $sp_{G,\lambda}(s,z) = sc_{G,\lambda}(s,z)$. So, if both paths between $s$ and $z$ are temporal paths, they must intersect at a timestep $\alpha$. But in this case, the set $\{\alpha\}$ would be a snapshot $s, z$-cut, again a contradiction. Therefore, we can suppose that there is no cycle containing $s$ and $z$. This means that, if we consider the decomposition of $G$ in bi-connected components $B_1, \ldots, B_k$, then we have that $s$ and $z$ are in different components. Moreover, as we are supposing $(G, \lambda), s, z$ that minimize $|V(G)| + |E(G)| + \tau(\lambda)$, the graph induced by the decomposition is a path and we can suppose that $s \in V(B_1)$ and $z \in V(B_k)$. Suppose that there is $i \in [k]$, such that $B_i$ contains at least 3 vertices. Let $v \in V(B_i) \cap V(B_j)$ for some $j \in \{i-1, i+1\}$ and $C_1$ be a cycle of $B_i$ containing $v$. If $B_j$ has at least 3 vertices, then we can find another cycle $C_2$ contained in $B_j$ such that $V(C_1) \cap V(C_2) = \{v\}$, this contradicts the fact that $G$ has no $M_3$ as m-topological minor. So, we can suppose that $|V(B_j)| = 2$. If the multiedge contained in $B_j$ has multiplicity 1, then let $\alpha$ be the timestep of such edge. We have that $\{\alpha\}$ is a snapshot $s, z$-cut, and then $sp_{G,\lambda}(s,z) = sc_{G,\lambda}(s,z)$ a contradiction. Therefore we can assume that the multiedge of $B_j$ has multiplicity at least 2; however the graph induced by such multiedge and $C_1$ is an m-subdivision of the graph $M_2$, again a contradiction. Thus, we can assume that bi-connected component $B_i$ contains a cycle, for every $i \in \{1, \ldots, k\}$. In other words, $U(G)$ is an $s, z$-path. If some of the multiedges of $G$ has multiplicity 1, then $sc_{G,\lambda}(s,z) \leq 1$ and we have the equality, a contradiction. We can say also that $|U(G)| = 3$, as otherwise $G$ would have $M_1$ as m-topological minor. Now, we show that in such case $G$ must be Mengerian, thus finishing the proof.

We show that, for a graph $H$ such that $U(H)$ is a path of size 3 between $s$ and $z$ and any timefunction $\lambda$ in $H$, we have that $sp_{G,\lambda}(s,z) = sc_{G,\lambda}(s,z)$. Let $V(H) = \{s, w, z\}$, which means that the multiedges of $H$ are $sw$ and $wz$. We use induction on the number of edges of $H$. The base of induction is when $sw$ and $wz$ both have multiplicity 1. Then, either $\lambda(sw) \leq \lambda(wz)$, in which case $sp_{H,\lambda}(s,z) = sc_{H,\lambda}(s,z) = 1$, or $\lambda(sw) > \lambda(wz)$, in which

case $sp_{H,\lambda}(s,z) = sc_{H,\lambda}(s,z) = 0$. Now, suppose valid when $|E(H')| \leq m$ and consider $|E(H)| = m+1$. By Proposition 11, we can suppose that there are two edges appearing at same timestep $\alpha$, say $f$ and $g$. As each snapshot is a simple graph, we get that $f, g$ form a temporal $s,z$-path. Let $H' = H - \{f,g\}$. By induction hypothesis, we have that $sp_{H',\lambda}(s,z) = sc_{H',\lambda}(s,z)$. Now let $\mathcal{P}$ be a set of snapshot disjoint temporal $s,z$-paths and $S$ be a snapshot $s,z$-cut in $(H',\lambda)$. Then, $\mathcal{P} \cup \{(s,f,w,g,z)\}$ is a set of snapshot disjoint temporal $s,z$-paths in $(H,\lambda)$ and $S \cup \{\alpha\}$ is a snapshot $s,z$-cut in $(H,\lambda)$. Therefore, $sc_{H,\lambda}(s,z) = sc_{H',\lambda}(s,z) + 1 = sp_{H',\lambda}(s,z) + 1 = sp_{H,\lambda}(s,z)$. ◀

Now we turn our attention to the recognition of Mengerian graphs, showing that it can be done in polynomial time. We observe that the proof of characterization of Mengerian graphs helps us to construct an algorithm of recognition of Mengerian graphs. We make the proper adaptation and prove the next theorem.

▶ **Theorem 12.** *One can decide in polynomial time whether a graph $G$ has a graph in $\mathcal{M}$ as m-topological minor.*

**Proof.** First, we find a decomposition of $G$ in 2-connected components, $B_1, \ldots, B_k$. This can be done in $O(m+n)$ (see e.g. [18]).

We consider the set of all components $B_i$ such that $|V(B_i)| = 2$ and the multiedge contained in $B_i$ has multiplicity at least 2, let $D$ be such set. Then, we test if some component $B_i$ in $D$ has vertex in common if a component $B_i$ of size at least 3. This takes $O(k^2)$ times. If the answer is positive, then we would have that $G$ has $M_2$ as m-topological minor. Then we can suppose the following:

**2.** No component in $D$ share a vertex with other component that has at least 3 vertices.

Now, we separate all components that have at least three vertices and test if two of them share a vertex, this step takes $O(k^2)$ times. If the answer is true for some two components, then $G$ has $M_3$ as m-topological minor. Therefore, we can suppose:

**3.** Components $B_i$ and $B_j$ of size at least 3 do not share vertex.

Now, we look to the components $B_i$ such that $|V(B_i)| \geq 4$. As $B_i$ is two connected, it has a cycle $C$ that we can find in $O(n^2)$. Then we check if there is an edge $e \in E(B_i) \setminus E(C)$. If the answer is positive, then as $B_i$ is 2-connected, there is a chord in $C$ containing $e$. This implies that $G$ has $M_5$ as m-topological minor. So, we have the following property.

**4.** Each component $B_i$ of size at least 4 is such that $U(B_i)$ is a cycle.

With such observation, we now can test if the components $B_i$ of size at least 4 contains some multiedge with multiplicity at least 2. If the answer is positive we are done as it would lead to $B_i$ having $M_4$ as m-topological minor. So, we can suppose otherwise:

**5.** Each component $B_i$ of size at least 4 contains no multiedges of multiplicity at least 2.

Now we show that the properties 2-5 implies that $G$ has no graphs in $\mathcal{M}$ as m-topological minors. One just need to observe that every cycle is contained in a 2-connected component. So the properties 2-5, assure us that $G$ has no $M_2$,$M_3$,$M_4$ or $M_5$ as m-topological minors.

Now, we only need to test if $G$ has $M_1$ as m-topological minor. We can consider the graph $G'$ obtained from $G$ but excluding the multiedges with multiplicity 1. Then, we test if $G'$ has a path with at least 4 vertices. The answer is positive if and only if $G$ has $M_1$ as m-topological minor. This finishes the recognition. ◀

## 6 Multiedge Cut

Finally, in this section we study another version of the cut problem. Recall that two temporal $s, z$-paths are multiedge disjoint if they do not share any multiedge, and that a multiedge temporal $s, z$-cut is a set $S$ of multiedges of $G$ intersecting every temporal $s, z$-path. In this section, we investigate the following problem.

$\leq h$-Multiedge Temporal s,z-cut
**Input.** A temporal graph $(G, \lambda)$, vertices $s, z \in V(G)$, and an integer $h$.
**Question.** Is there a multiedge temporal $s, z$-cut in $(G, \lambda)$ of size at most $h$?

▶ **Theorem 13.** *$\leq h$-Multiedge Temporal s,z-cut is* NP*-complete, even if $\tau = 2$.*

**Proof.** We make a reduction from Vertex Cover, which consists of, given a simple graph $G$ and a positive integer $k$, deciding whether there exists a subset $S \subseteq V(G)$ such that $|S| \leq k$ and every $e \in E(G)$ is incident to some $u \in S$; such a set is called a *vertex cover* (of size at most $k$). So, consider $\mathcal{I} = (G, k)$ an instance of Vertex Cover. We construct a graph $G'$ with vertex set $V(G') = \{s, z\} \cup \{x_v^1, x_v^2, x_v^3, x_v^4 \colon v \in V(G)\} \cup \{f_{vw} \mid vw \in E(G)\}$. One can use Figure 9 to follow the construction. Then, we add edges from $s$ to $x_v^1$ and $x_v^2$, and from $x_v^3$ and $x_v^4$ to $z$, for every $v \in V(G)$. Also, let $(x_v^1, x_v^2, x_v^3, x_v^4)$ form a path, and add, for each edge $vw \in E(G)$, edges $x_v^1 f_{vw}$ and $f_{vw} x_w^4$. More formally, we have:

$$\begin{aligned} E(G') = \quad & \{s x_v^1, s x_v^2, x_v^3 z, x_v^4 z \mid v \in V(G)\} \\ & \cup \{x_v^i x_v^{i+1} \mid i \in \{1,2,3\}, v \in V(G)\} \\ & \cup \{x_v^1 f_{vw}, f_{vw} x_w^4 \mid vw \in E(G)\} \end{aligned} .$$

Finally, add a second edge with endpoints $x_v^2 x_v^3$, for each $v \in V(G)$. Since these are the only edges with multiplicity greater than 1, we will generally denote an edge by its endpoints, with the exception of these, which we denote by $e_v^1, e_v^2$. Now for each $i \in \{1, \ldots, 4\}$ define $X_i = \{x_v^i \colon v \in V\}$, let $F = \{f_{vw} \mid vw \in E(G)\}$ and consider a timefunction $\lambda$ such that:

$$\lambda(e) = \begin{cases} 1 & \text{, if } e \in (\{s\}) \times X_1) \cup (X_1 \times X_2) \cup (X_3 \times \{z\}) \cup (X_1 \times F) \\ 2 & \text{, if } e \in (\{s\} \times X_2) \cup (X_3 \times X_4) \cup (X_4 \times \{z\}) \cup (F \times X_4), \text{ and} \\ i & \text{, if } e = e_v^i \text{ for some } v \in V(G). \end{cases}$$

We prove that $G$ has a vertex cover of size at most $k$ if and only if $(G', \lambda)$ has a multiedge temporal $s, z$-cut of size at most $n + k$. Given a solution $S$ of Vertex Cover, we can define $S' = \{s x_v^1, x_v^4 z \colon v \in S\} \cup \{x_v^2 x_v^3 \colon v \notin S\}$. It remains to argue that $S'$ separates $s$ from $z$, or more formally, that there is no temporal $s, z$-path in $(G' - S^*, \lambda)$, where $S^*$ contains every $e \in E(G')$ such that the endpoints of $e$ are in $S'$. Notice that if a temporal $s, z$-path does not use edges between $X_1$ and $F$, then it contains one of the following paths: $(s, x_v^1, x_v^2, x_v^3, x_v^4, z)$, $(s, x_v^2, x_v^3, x_v^4, z)$, or $(s, x_v^1, x_v^2, x_v^3, z)$. In any case such path intersects $S'$. Now suppose that a temporal $s, z$-path, $P$, uses an edge $x_v^1 f_{vw}$, which implies that it also uses $f_{vw} x_w^4$ and therefore it arrives in $z$ at timestep 2. Note that the only edges active at timeste 2 incident to $x_w^4$ are incident either to $F$, or to $x_w^3$, or to $z$. In the former case, we hit a dead end because $f_{uw}$ has only one edge incident to it in timestep 2, for every $u \in N(w)$. This also happens in the second case, since from $x_w^3$ one can only go to $x_w^2$, hitting again a dead end. We then get that if $P$ contains $x_v^1 f_{vw}$, then it must also contain $x_w^4 z$. By a similar argument one can also show that it must contain $s x_v^1$ too. As $vw \in E(G)$, at least one of $v$ and $w$ are in $S$, and by construction we get that $P$ uses some multiedge of $S'$, as we wanted to show.

Now let $S'$ be a multiedge temporal $s, z$-cut of size at most $n + k$. For each $v \in V(G)$, let the set of multiedges $\{s x_v^1, x_v^1 x_v^2, x_v^2 x_v^3, x_v^3 x_v^4, x_v^4 z\}$ be denoted by $A_v$, and let $S'_v = S' \cap A_v$.

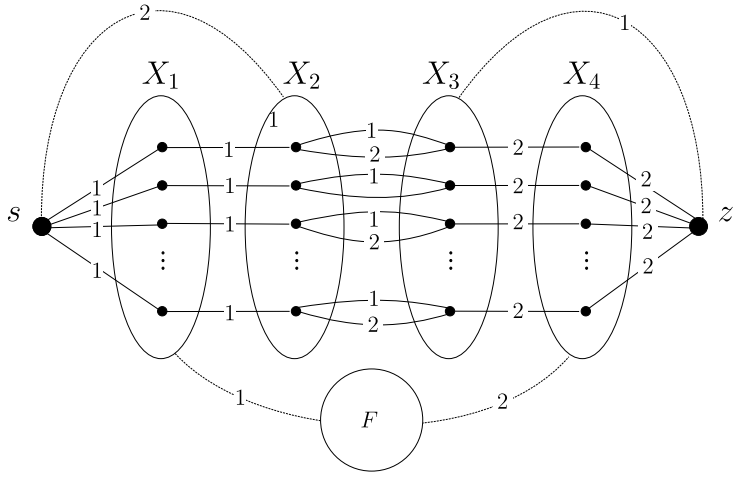



**Figure 9** A representation of temporal graph $(G', \lambda)$. The dotted lines represent all the edges between the corresponding set of vertices.

Because $A_v$ forms a temporal $s, z$-path, we know that $S'_v \neq \emptyset$. One can notice that every temporal $s, z$-path using $x_v^1 x_v^2$ also uses $sx_v^1$, and in the same way every temporal $s, z$-path using $x_v^3 x_v^4$ uses $x_v^4 t$. Then by changing $S'$ if necessary, we can suppose that $S'_v$ is a non-empty subset of $\{sx_v^1, x_v^2 x_v^3, x_v^4 z\}$. Now we show that we can actually suppose that $S' \cap A_v$ is either $\{sx_v^1, x_v^4 t\}$ or $\{x_v^2 x_v^3\}$. We do it by analysing the cases where this does not happen.

- $S'_v = \{sx_v^1\}$ or $S'_v = \{x_v^4 z\}$. We just solve the first subcase as the second is similar. Notice that $(s, x_v^2, x_v^3, x_v^4, z)$ is a temporal $s, z$-path, and that the only edge in this path that can be in $S'$ is $sx_v^2$ because of the case being analyzed. So, removing $sx_v^2$ and adding $x_v^4$ to $S'$ maintains the property of being a multiedge temporal $s, z$-cut and turns $S'_v$ into $\{sx_v^1, x_v^4 z\}$.
- $S'_v = \{sx_v^1, x_v^2 x_v^3\}$ or $S'_v = \{x_v^2 x_v^3, x_v^4 z\}$. In both subcases we can remove $x_v^2 x_v^3$ and add either $x^4 z$ (in the first case) or $sx_v^1$ (in the second one).

Now we can define $S = \{v \in V : S_v = \{sx_v^1, x_v^4 t\}\}$. The desired property $|S| \leq k$ follows from the fact that $1 \leq |S'_v| \leq 2$ for every $v \in V(G)$, and that $|S'| \leq n + k$. Finally, suppose that $vw \in E(G)$ is such that $S \cap \{v, w\} = \emptyset$. Then $(s, x_v^1, x_w^4, z)$ is a temporal $s, z$-path not passing through the edges in $S'$, a contradiction. ◀

## 7 Conclusion

We have introduced the concept of snapshot disjointness and proved that the related paths and cut problems, when parameterized by the size of the solution, are both W[1]-hard and XP-time solvable. We then adapted to our context the definition of Mengerian graph introduced by Kempe, Kleinberg and Kumar [13], giving also a characterization in the lines of the ones given in [13] and [11], as well as a polynomial-time recognition algorithm. Since all our results concern only non-strict temporal paths, one can ask whether they also hold for strict paths.

Further open problems can be extracted from Table 1. In particular, we ask whether the results for node departure disjoint paths and cuts presented in [16] for strict paths also hold for non-strict paths.

Finally, while Menger's Theorem is known to hold for edge disjoint paths [3, 16], it is also known not to hold for multiedge disjoint paths [11]. We reinforce the question posed in [11] about the characterization of Mengerian graphs in the context of multiedge disjoint paths.

## References


1 Amir Afrasiabi Rad, Paola Flocchini, and Joanne Gaudet. Computation and analysis of temporal betweenness in a knowledge mobilization network. *Computational social networks*, 4(1):1–22, 2017.

2 Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondřej Pangrác, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. *ACM Transactions on Algorithms (TALG)*, 7(1):1–27, 2010.

3 Kenneth A Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks: An International Journal*, 28(3):125–134, 1996.

4 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

5 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. In *31st International Symposium on Algorithms and Computation, ISAAC 2020*, volume 181 of *LIPIcs*, pages 30:1–30:18, 2020. doi:10.4230/LIPIcs.ISAAC.2020.30.

6 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5(4). Springer, 2015.

7 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Temporal Connectivity: Coping with Foreseen and Unforeseen Delays. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*, volume 221 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2022.17.

8 Petr A Golovach and Dimitrios M Thilikos. Paths of bounded length and their cuts: Parameterized complexity and algorithms. *Discrete Optimization*, 8(1):72–86, 2011.

9 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.

10 Allen Ibiapina, Raul Lopes, Andrea Marino, and Ana Silva. Menger's theorem for temporal paths (not walks). *ArXiv*, 2022. arXiv:2206.15251.

11 Allen Ibiapina and Ana Silva. Mengerian graphs: Characterization and recognition. *arxiv*, 2022. arXiv:2208.06517.

12 Alon Itai, Yehoshua Perl, and Yossi Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982.

13 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64:820–842, 2002.

14 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1):61, 2018.

15 Chung-Lun Li, S Thomas McCormick, and David Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics*, 26(1):105–115, 1990.

16 George B Mertzios, Othon Michail, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.

17 Yossi Shiloach and Yehoshua Perl. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the ACM (JACM)*, 25(1):1–9, 1978.

**18** Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

**19** B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

**20** Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *J. Comput. Syst. Sci.*, 107:72–92, 2020. doi:10.1016/j.jcss.2019.07.006.

# Partial Gathering of Mobile Agents in Dynamic Tori

**Masahiro Shibata**
Kyushu Institute of Technology, Fukuoka, Japan

**Naoki Kitamura**
Osaka University, Japan

**Ryota Eguchi**
NAIST, Nara, Japan

**Yuichi Sudo**
Hosei University, Tokyo, Japan

**Junya Nakamura**
Toyohashi University of Technology, Aichi, Japan

**Yonghwan Kim**
Nagoya Institute of Technology, Aichi, Japan

## Abstract

In this paper, we consider the partial gathering problem of mobile agents in synchronous dynamic tori. The partial gathering problem is a generalization of the (well-investigated) total gathering problem, which requires that all $k$ agents distributed in the network terminate at a non-predetermined single node. The partial gathering problem requires, for a given positive integer $g\,(< k)$, that agents terminate in a configuration such that either at least $g$ agents or no agent exists at each node. So far, in almost cases, the partial gathering problem has been considered in static graphs. As only one exception, it is considered in a kind of dynamic rings called 1-interval connected rings, that is, one of the links in the ring may be missing at each time step. In this paper, we consider partial gathering in another dynamic topology. Concretely, we consider it in $n \times n$ dynamic tori such that each of row rings and column rings is represented as a 1-interval connected ring. In such networks, when $k = O(gn)$, focusing on the relationship between the values of $k, n$, and $g$, we aim to characterize the solvability of the partial gathering problem and analyze the move complexity of the proposed algorithms when the problem can be solved. First, we show that agents cannot solve the problem when $k = o(gn)$, which means that $\Omega(gn)$ agents are necessary to solve the problem. Second, we show that the problem can be solved with the total number of $O(gn^3)$ moves when $2gn + 2n - 1 \leq k \leq 2gn + 6n + 16g - 12$. Finally, we show that the problem can be solved with the total number of $O(gn^2)$ moves when $k \geq 2gn + 6n + 16g - 11$. From these results, we show that our algorithms can solve the partial gathering problem in dynamic tori with the asymptotically optimal number $\Theta(gn)$ of agents. In addition, we show that agents require a total number of $\Omega(gn^2)$ moves to solve the partial gathering problem in dynamic tori when $k = \Theta(gn)$. Thus, when $k \geq 2gn + 6n + 16g - 11$, our algorithm can solve the problem with asymptotically optimal number $O(gn^2)$ of agent moves.

**2012 ACM Subject Classification** Theory of computation → Self-organization

**Keywords and phrases** distributed system, mobile agents, partial gathering, dynamic tori

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.2

**Funding** This work was partially supported by JSPS KAKENHI Grant Number 18K18031, 20H04140, 20KK0232, 21K17706, and 22K11971; and Foundation of Public Interest of Tatematsu.

© Masahiro Shibata, Naoki Kitamura, Ryota Eguchi, Yuichi Sudo, Junya Nakamura, and Yonghwan Kim;
licensed under Creative Commons License CC-BY 4.0



# 1 Introduction

## 1.1 Background and Related Work

A *distributed system* comprises a set of computing entities (*nodes*) connected by communication links. As a promising design paradigm of distributed systems, (mobile) agents have attracted much attention [7]. The agents can traverse the system, carrying information collected at visited nodes, and execute an action at each node using the information to achieve a task. In other words, agents can encapsulate the process code and data, which simplifies the design of distributed systems [10].

The *total gathering problem* (or the rendezvous problem) is a fundamental problem for agents' coordination. When a set of $k$ agents are arbitrarily placed at nodes, this problem requires that all the $k$ agents terminate at a non-predetermined single node. By meeting at a single node, all agents can share information or synchronize their behaviors. The total gathering problem has been considered in various kinds of networks such as rings [9, 13], trees [5, 1], tori [8], and arbitrary networks [3, 4].

Recently, a variant of the total gathering problem, called the *g-partial gathering problem* [14], has been considered. This problem does not require all agents to meet at a single node, but allows agents to meet at several nodes separately. Concretely, for a given positive integer $g$ ($< k$), this problem requires that agents terminate in a configuration such that either at least $g$ agents or no agent exists at each node. Notice that the $g$-partial gathering problem is equivalent to the total gathering problem when $k < 2g$. From a practical point of view, the $g$-partial gathering problem is still useful especially in large-scale networks. That is, when $g$-partial gathering is achieved, agents are partitioned into groups each of which has at least $g$ agents, each agent can share information and tasks with agents in the same group, and each group can partition the network and then patrol its area that it should monitor efficiently.

As related work, Shibata et al. considered the $g$-partial gathering problem in rings [14, 15, 20], trees [17], and arbitrary networks [16]. In [14, 15], they considered it in unidirectional ring networks with whiteboards (or memory spaces that agents can read and write) at nodes. They mainly showed that, if agents have distinct IDs and the algorithm is deterministic, or if agents do not have distinct IDs and the algorithm is randomized, agents can achieve $g$-partial gathering with the total number of $O(gn)$ moves (in expectation), where $n$ is the number of nodes. In [20], they considered $g$-partial gathering for another mobile entity called *mobile robots* that have no memory but can observe all nodes and robots in the network. In the case of using mobile robots, they also showed that $g$-partial gathering can be achieved with the total number of $O(gn)$ moves. In addition, the $g$-partial (resp., the total) gathering problem in ring networks requires a total number of $\Omega(gn)$ (resp., $\Omega(kn)$) moves in both agent and robot models. Thus, the above results are asymptotically optimal in terms of the total number of moves, and the number $O(gn)$ is strictly smaller than that for the total gathering problem when $g = o(k)$. In tree and arbitrary networks, they also proposed algorithms to solve the $g$-partial gathering problem with strictly smaller total number of moves compared to the total gathering problem for some settings.

While all the above work on the total gathering problem and the $g$-partial gathering problem are considered in *static graphs* where a network topology does not change during an execution, recently many problems involving agents have been studied in *dynamic graphs*, where a topology changes during an execution. For example, the total gathering problem [12], the exploration problem [11, 6], the compacting and grouping problem [2], and the uniform deployment problem [18] are considered in dynamic graphs. Also, in [19], the $g$-partial

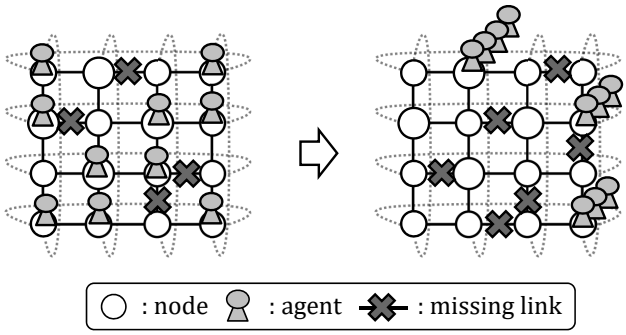



**Figure 1** An example of the $g$-partial gathering problem in a $4 \times 4$ dynamic torus ($g = 3$).

gathering problem is considered in a kind of dynamic rings called *1-interval connected rings* [12, 11, 18], that is, one of the links in the ring may be missing at each time step. In such networks, focusing on the relationship between the values of $k$ and $g$, they clarified the solvability of the $g$-partial gathering problem and analyzed the move complexity of the proposed algorithms when the problem can be solved. As a result, they showed that (i) when $k \leq 2g$, the $g$-partial gathering problem cannot be solved, (ii) when $2g + 1 \leq k \leq 3g - 2$, the problem can be solved with the total number of $O(gn \log g)$ moves, and (iii) when $k \geq 3g - 1$, the problem can be solved with the asymptotically optimal total number $O(gn)$ of agent moves.

## 1.2 Our Contribution

In this paper, we consider the $g$-partial gathering problem of mobile agents in another dynamic topology. Concretely, we consider the problem in $n \times n$ dynamic tori such that each of row rings and column rings is represented as a 1-interval connected ring. An example is given in Fig. 1. An edge with a cross means that it is missing. In this paper, we assume that each node has a whiteboard. In addition, we assume that agents have distinct IDs, common sense of direction, knowledge of $k$ and $n$, and behave fully synchronously. In such settings, when $k = O(gn)$, focusing on the relationship between the values of $k, n$, and $g$, we aim to characterize the solvability of the $g$-partial gathering problem and analyze the time and move complexities of the proposed algorithms when the problem can be solved.

We summarize our results in Table 1. First, we show that agents cannot solve the $g$-partial gathering problem when $k = o(gn)$, which means that $\Omega(gn)$ agents are necessary to solve the problem. Second, we show that the problem can be solved with $O(n^2)$ rounds and the total number of $O(gn^3)$ moves when $2gn + 2n - 1 \leq k \leq 2gn + 6n + 16g - 12$. Finally, we show that the problem can be solved with $O(n^2)$ rounds and the total number of $O(gn^2)$ moves when $k \geq 2gn + 6n + 16g - 11$. From these results, we show that our algorithms can solve the $g$-partial gathering problem in dynamic tori with the asymptotically optimal number $\Theta(gn)$ of agents. In addition, we show that agents require a total number of $\Omega(gn^2)$ moves to solve the $g$-partial gathering problem in $n \times n$ dynamic tori when $k = \Theta(gn)$. Thus, when $k \geq 2gn + 6n + 16g - 11$, our algorithm can solve the problem with asymptotically optimal number $O(gn^2)$ of agent moves.

Due to the page limitation, we omit to describe several pseudocodes and several proofs of theorems and lemmas.

**Table 1** Results of $g$-partial gathering for agents with distinct IDs in dynamic tori when $k = O(gn)$ ($n$: #nodes, $k$: #agents).

| | Result 1 (Sec. 3) | Result 2 (Sec. 5) | Result 3 (Sec. 6) |
|---|---|---|---|
| Relation between $k$ and $g$ | $k = o(gn)$ | $k \geq 2gn + 2n - 1$ and $k \leq 2gn + 6n + 16g - 12$ | $k \geq 2gn + 6n + 16g - 11$ |
| Solvable/Unsolvable | Unsolvable | Solvable | Solvable |
| Time complexity | - | $O(n^2)$ | $O(n^2)$ |
| Total number of agent moves | - | $O(gn^3)$ | $\Theta(gn^2)$ |

# 2 Preliminaries

## 2.1 System Model

We basically follow the model defined in [6]. An $n \times n$ *dynamic torus* $T$ is defined as 2-tuple $T = (V, E)$, where $V$ is a set of nodes $\{v_{i,j} \mid 0 \leq i, j \leq n-1\}$ and $E$ is a set of links $\{(v_{i,j}, v_{(i+1) \mod n,j}), (v_{i,j}, v_{i,(j+1) \mod n}) \mid 0 \leq i, j \leq n-1\}$. For simplicity, we denote $v_{(i+i') \mod n,(j+j') \mod n}$ by $v_{i+i',j+j'}$ for any integers $i, i', j$, and $j'$. The *distance* between nodes $v_{i,j}$ and $v_{p,q}$ is defined as $\min\{i-p, p-i\} + \min\{j-q, q-j\}$. Notice that this definition of the distance is correct when no corresponding link that connects $v_{i,j}$ and $v_{p,q}$ is missing. We call the direction from $v_{i,j}$ to $v_{i,j+1}$ (resp., to $v_{i+1,j}$, to $v_{i,j-1}$, and to $v_{i-1,j}$) the *right* (resp., *down*, *left*, and *up*) direction. Intuitively, torus $T$ comprises $n$ row rings and $n$ column rings. A row ring $R_i$ (resp., a column ring $C_j$) is a subgraph of $T$ induced by $\{v_{i,j} \mid 0 \leq j \leq n-1\}$ (resp., $\{v_{i,j} \mid 0 \leq i \leq n-1\}$) (see Fig. 2). We assume that each of row rings and column rings is *1-interval connected*, that is, one of the links in the ring may be missing at each time step, and which link is missing is controlled by an *adversarial scheduler*. Then, since each of the row and column rings is 1-interval connected, the dynamic torus $T$ is always connected. In addition, we assume that nodes are anonymous, i.e., they do not have IDs (and thus the indices of nodes are used just for notation purposes). Every node $v_{i,j} \in V$ has a whiteboard that agents at node $v_{i,j}$ can read from and write on.

Let $A = \{a_0, a_1, \ldots, a_{k-1}\}$ be a set of $k\,(\leq n)$ agents. Agents can move through links, that is, they can move from $v_{i,j}$ to $v_{i,j+1}$ (move right), from $v_{i,j}$ to $v_{i+1,j}$ (move down), from $v_{i,j}$ to $v_{i,j-1}$ (move left), or from $v_{i,j}$ to $v_{i-1,j}$ (move up), for any $i$ and $j$. Agents have distinct IDs and knowledge of $k$ and $n$. Agents have the common sense of directions, that is, they agree on the directions of right, down, left, and up in the torus. In addition, agents cannot detect whether other agents exist at the current node or not. An agent $a_h$ is defined as a deterministic finite automaton ($S$, $W$, $\delta$, $s_{initial}$, $s_{final}$, $w_{initial}$, $w'_{initial}$). The first element $S$ is the set of all states of an agent, including two special states, initial state $s_{initial}$ and final state $s_{final}$. The second element $W$ is the set of all states (contents) of a whiteboard, including two special initial states $w_{initial}$ and $w'_{initial}$. We explain $w_{initial}$ and $w'_{initial}$ in the next paragraph. The third element $\delta : S \times W \mapsto S \times W \times M$ is the state transition function that decides, from the current state of $a_h$ and the current node's whiteboard, the next states of $a_h$ and the whiteboard, and whether $a_h$ moves to its neighboring node or not. The last element $M$ ={null, right, down, left, up} in $\delta$ represents which direction $a_h$ tries to move in the next movement. The value "null" means staying at the current node. We assume that $\delta(s_{final}, w_{ij}) = (s_{final}, w_{ij}, \text{null})$ holds for any state $w_{ij} \in W$, which means that $a_h$ never changes its state, updates the contents of the current node $v_{i,j}$'s whiteboard, or leaves $v_{i,j}$ once it reaches state $s_{final}$. We say that an agent *terminates* when its state changes to $s_{final}$. Notice that $S, \delta, s_{initial}$, and $s_{final}$ can be dependent on the agent's ID.

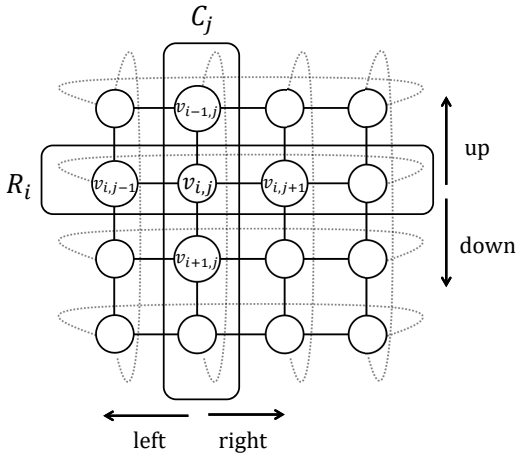



**Figure 2** An example of a torus graph ($n = 4$).

In an agent system, (global) *configuration* $c$ is defined as a product of the states of all agents, the states (whiteboards' contents) of all nodes, and the locations (i.e., the current nodes) of all agents. We define $C$ as a set of all configurations. In an initial configuration $c_0 \in C$, we assume that agents are deployed arbitrarily at mutually distinct nodes (or no two agents start at the same node), and the state of each whiteboard is $w_{initial}$ or $w'_{initial}$ depending on the existence of an agent. That is, when an agent exists at node $v_{i,j}$ in the initial configuration, the initial state of $v'_{i,j}$s whiteboard is $w_{initial}$. Otherwise, the state is $w'_{initial}$.

During an execution of the algorithm, we assume that agents move instantaneously, that is, agents always exist at nodes (do not exist on links). Each agent executes the following four operations in an *atomic action*: 1) reads the contents of its current node's whiteboard, 2) executes local computation (or changes its state), 3) updates the contents of the current node's whiteboard, and 4) moves to its neighboring node or stays at the current node. If several agents exist at the same node, they take atomic actions interleavingly in an arbitrary order. In addition, when an agent tries to move to its neighboring node (e.g., from node $v_{i,j}$ to $v_{i,j+1}$) but the corresponding link is missing, we say that the agent is *blocked*, and it still exists at $v_{i,j}$ at the beginning of the next atomic action.

In this paper, we consider a *synchronous execution*, that is, in each time step called *round*, all agents perform atomic actions. Then, an *execution* starting from $c_0$ is defined as $E = c_0, c_1, \ldots$ where each $c_i$ ($i \geq 1$) is the configuration reached from $c_{i-1}$ by atomic actions of all agents. An execution is infinite, or ends in a *final configuration* where the state of every agent is $s_{final}$.

## 2.2 The Partial Gathering Problem

The requirement for the partial gathering problem is that, for a given integer $g$, agents terminate in a configuration such that either at least $g$ agents or no agent exists at each node. Formally, we define the $g$-partial gathering problem as follows.

▶ **Definition 1.** *An algorithm solves the $g$-partial gathering problem in dynamic tori when the following conditions hold:*

- *Execution $E$ is finite (i.e., all agents terminate in state $s_{final}$).*
- *In the final configuration, at least $g$ agents exist at any node where an agent exists.*

In this paper, we evaluate the proposed algorithms by the time complexity (the number of rounds required for agents to solve the problem) and the total number of agent moves.

## 3 The case of $k = o(gn)$

When $k = o(gn)$, the following impossibility result holds. Intuitively, this is because (i) when there exists an agent at each of nodes $v_{0,0}, v_{1,1}, \ldots, v_{n-1,n-1}$ in the initial configuration, the adversary can make the $n$ agents never leave their starting nodes, and thus (ii) to achieve $g$-partial gathering, it is necessary that there exist at least $g$ agents at each of the $n$ nodes, which requires at least $gn$ agents in total.

▶ **Theorem 2.** *When $k = o(gn)$ holds, the $g$-partial gathering problem cannot be solved in dynamic tori.*

## 4 Lower bound on the total number of agent moves when $k = \Theta(gn)$

By Theorem 3, since at least $\Omega(gn)$ agents are necessary to solve the problem, in the following, we assume that there exist $\Theta(gn)$ agents in the torus. Then, we have the following theorem on the lower bound of the total number of agent moves. Intuitively, this is because when there exists an agent at each of nodes $v_{0,0}, v_{1,1}, \ldots, v_{n-1,n-1}$ and each of the other agents is placed at a node with distance $\Omega(n)$ from $v_{i,i}$ $(0 \leq i \leq n-1)$ in the initial configuration, and when the adversary makes agents at $v_{i,i}$ never leave their starting node, it is necessary that at least $gn - n$ agents need to stay at either $v_{0,0}, v_{1,1}, \ldots,$ or $v_{n-1,n-1}$, which requires the total number of $\Omega(gn^2)$ moves in total.

▶ **Theorem 3.** *A lower bound on the total number of agent moves to solve the $g$-partial gathering problem in dynamic tori when $k = \Theta(gn)$ is $\Omega(gn^2)$.*

## 5 The case of $2gn + 2n - 1 \leq k \leq 2gn + 6n + 16g - 12$

In this section, when $2gn + 2n - 1 \leq k \leq 2gn + 6n + 16g - 12$, we propose an algorithm to solve the $g$-partial gathering problem with $O(n^2)$ rounds and the total number of $O(gn^3)$ moves. In the algorithm, agents aim to make a configuration such that there exist at least $2g + 1$ agents in each of row rings. Then, agents achieve $g$-partial gathering by applying the existing method [19] for $g$-partial gathering in 1-interval connected rings to each row ring independently. To this end, agents repeat the following two phases $n$ times: the counting phase and the adjusting phase. In the counting phase, each agent in row ring $R_i$ tries to move horizontally to count the number of agents existing in $R_i$ at the beginning of the counting phase. In the adjusting phase, several agents in row rings with a lot of agents try to move vertically to a row ring with less agents at the beginning of the adjusting phase.

The overall pseudocode is given in Algorithm 1 (the pseudocodes of the counting phase and the adjusting phase are given as procedures *Counting()* and *Adjusting()*, respectively). Global variables used in the algorithm are given in Table 2. In the following subsections, we explain the details of each phase.

### 5.1 Counting phase

The aim of this phase is that each agent $a_h$ in row ring $R_i$ calculates the total number of agents existing in $R_i$ at the beginning of this counting phase by making either of the following two configurations: (i) Each agent $a_h$ travels once around the row ring $R_i$ and gets IDs of

**Table 2** Global variables used in proposed algorithms.

**Variables for agent $a_h$.**

| Type | Name | Meaning | Initial value |
|---|---|---|---|
| int | $a_h.phase$ | phase number stored by $a_h$ | 0 or 1 |
| int | $a_h.rounds$ | number of rounds from some round | 1 |
| int | $a_h.nVisited$ | number of nodes that $a_i$ has visited from some round | 0 |
| int | $a_h.nAgentsRowRing$ | number of agents existing in the current row ring | 0 |
| int | $a_h.rank$ | ordinal number of how its ID is small among IDs of agents at the same node | 0 |
| int | $a_h.dir$ | direction to which $a_h$ tries to move (1: right or down, -1: left or up ) | 0 |
| array | $a_h.IDs[p]$ | list of IDs that $a_h$ has observed in the phase $p$ | $\perp$ |

**Variables for node $v_{i,j}$.**

| Type | Name | Meaning | Initial value |
|---|---|---|---|
| int | $v_{ij}.phase$ | phase number stored by $v_{i,j}$ | 1 |
| int | $v_{ij}.nAgentsCurrent$ | number of agents staying at the current node $v_{i,j}$ | 0 |
| array | $v_{ij}.IDs[p]$ | list of IDs stored by $v_{i,j}$ in phase $p$ | $\perp$ |
| array | $v_{ij}.nAgentsAdjust[p]$ | number of agents existing in the current row ring at the beginning of the current adjusting phase with phase number $p$ | $\perp$ |

**Algorithm 1** The behavior of agent $a_h$ for the proposed algorithm when $2gn + 2n - 1 \leq k \leq 2gn + 6n + 16g - 12$.

**Main Routine of Agent $a_h$:**
1 $a_h.phase := 1$
2 **while** $a_h.phase \leq n$ **do**
3  *Counting()*
4  *Adjusting()*
5  $a_h.phase{+}{+}$
6 Apply the existing method [19] to the currently staying row ring
7 Terminate the algorithm execution

all the agents existing in $R_i$, or (ii) it detects that all the agents existing in $R_i$ are at the same node. To this end, we use an idea similar to [12] which considers total gathering in 1-interval connected rings. First, each agent $a_h$ writes its ID $a_h.id$ and the current phase number $a_h.phase$ to the variables $v_{ij}.IDs[a_h.phase]$ and $v_{ij}.phase$, respectively, on the current node $v_{i,j}$'s whiteboard and then tries to move right for $3n$ rounds. During the movement, $a_h$ memorizes values of observed IDs that are written in the current counting phase to array $a_h.IDs[a_h.phase]$. After the $3n$ rounds, the number $a_h.nVisited$ of nodes that $a_h$ has visited from the beginning of this counting phase is (a) at least $n$ or (b) less than $n$ due to missing links. In case (a), $a_h$ must have completed traveling once around the row ring $R_i$. Hence, $a_h$ can calculate the number $a_h.nAgentsRowRing$ of agents existing in $R_i$ through the number of observed IDs (i.e., $|a_h.IDs[a_h.phase]|$). Thus, it reaches configuration (i). In case (b) (i.e., $a_h$ has visited less than $n$ nodes during the $3n$ rounds), we show in Lemma 4 that all the agents existing in $R_i$ stay at the same node (they reach configuration (ii)). Thus, through the value $v_{ij}.nAgentsCurrent$ of the current node $v'_{i,j}s$ whiteboard representing the number of agents currently staying at $v_{i,j}$, $a_h$ can calculate $a_h.nAgentsRowRing$.

**Algorithm 2** Procedure *Counting()* ($v_{i,j}$ is the current node of $a_h$).

**Main Routine of Agent $a_h$:**

1 $v_{ij}.phase := a_h.phase, v_{ij}.IDs[v_{ij}.phase] := v_{ij}.IDs[v_{ij}.phase] \cup a_h.id, v_{ij}.nAgentsCurrent{+}{+}$
2 $a_h.nVisited := 1$, $a_h.rounds := 1$, $a_h.IDs[a_h.phase] := v_{ij}.IDs[v_{ij}.phase]$
3 **while** $a_h.rounds \leq 3n$ **do**
4  $v_{ij}.nAgentsCurrent$--
5  Try to move from the current node $v_{i,j}$ to the right neighboring node $v_{i,j+1}$
6  **if** $a_h$ *reached* $v_{i,j+1}$ *(that becomes new* $v_{i,j}$*)* **then**
7  $a_h.nVisited{+}{+}$
8  **if** $(v_{ij}.phase = a_h.phase) \wedge (a_h.nVisited \leq n)$ **then**
9  $a_h.IDs[a_h.phase] := a_h.IDs[a_h.phase] \cup v_{ij}.IDs[v_{ij}.phase]$
10  $v_{ij}.nAgentsCurrent{+}{+}$, $a_h.rounds{+}{+}$
11 **if** $a_h.nVisited < n$ **then** $a_h.nAgentsRowRing := v_{ij}.nAgentsCurrent$ // all the agents in row ring $R_i$ stay at the current node
12 **if** $a_h.nVisited \geq n$ **then** $a_h.nAgentsRowRing := |a_h.IDs[a_h.phase]|$ // $a_h$ traveled once around the row ring
13 Terminate the counting phase and enter the adjusting phase

The pseudocode of the counting phase is described in Algorithm 2. Note that, during the counting phase, an agent $a_h$ may visit more than $n$ nodes and then the number of observed IDs is more than the number of agents in the row ring when it is blocked less than $2n$ times. In this case, $a_h$ stops measuring IDs when it has visited more than $n$ nodes (line 8).

Concerning the counting phase, we have the following lemma.

▶ **Lemma 4.** *Let $na_i$ be the number of agents existing in row ring $R_i$ at the beginning of the current counting phase with phase number $p$. Then, at the end of the counting phase, each agent $a_h$ existing in $R_i$ in phase $p$ stores the correct value of $na_i$ to $a_h.nAgentsRowRing$.*

## 5.2 Adjusting phase

In this phase, several agents in a row ring with a lot of agents try to move vertically to a row ring with few agents to reduce the gap of the number of agents between row rings. Concretely, several agents in each row ring $R_i$ first move horizontally in $R_i$ and write the number of agents existing in $R_i$ at the beginning of this adjusting phase (i.e., $a_h.nAgentsRowRing$ for agent $a_h$), to each node's whiteboard of $R_i$. Then, several agents belonging to a row ring with at least $2g + 3$ agents try to move in the torus vertically and stay at a node of a row ring with less than $2g + 1$ agents at the beginning of this adjusting phase. By repeating this behavior and the counting phase explained before, agents eventually reach a configuration such that there exist at least $2g + 1$ agents in each row ring (and then $g$-partial is achieved by applying the existing method [19] to each row ring independently).

First, we explain how to write the number $na_i$ of agents in $R_i$ at the beginning of this adjusting phase (or at the end of the counting phase just before) to each node's whiteboard of $R_i$. By Lemma 4, at the end of the counting phase just before, each agent $a_h$ in row ring $R_i$ knows the number $na_i$ ($= a_h.nAgentsRowRing$) of agents currently existing in $R_i$. In addition, by Algorithm 2, $a_h$ can get the list of agent IDs existing in $R_i$. Among the agents, let $a_1^i$ (resp., $a_2^i$) be the agent with the smallest (resp., the second smallest) ID. Then, for $n$ rounds, $a_1^i$ (resp., $a_2^i$) tries to move right (resp., left) and then $a_1^i$ (resp., $a_2^i$) tries to move left (resp., right) for the next $n$ rounds. During the movement, $a_1^i$ and $a_2^i$ write values of $na_i$ for the current phase $p$ to variable $v_{ij}.nAgentsAdjust[p]$ of each node $v_{i,j}$'s whiteboard. By this

behavior, we show in Lemma 5 that every node in $R_i$ is visited by $a_1^i$ or $a_2^i$ and the value of $na_i$ is stored when there exist at least two agents in $R_i$. Notice that it is possible that there exists only one agent $a$ in $R_i$ at the beginning of some adjusting phase and $a$ cannot visit all nodes in $R_i$ and write necessary information to nodes' whiteboards in $R_i$. In this case, when some agents in a row ring other than $R_i$ try to move vertically and visit some node in $R_i$ (this method is explained in the next paragraphs), they can recognize that no information is written to the node of $R_i$ and there exist less than $2g+1$ (actually one) agents in $R_i$.

Next, we explain how agents in a row ring with many agents move vertically to a node of a row ring with less agents. First, when $na_i \geq 2g+3$, for $3n$ rounds, each agent $a_h$ in $R_i$ tries to move right until reaching the node $v_{min}^i$ where the smallest ID is written in the counting phase just before. Then, by the similar discussion of agents' behaviors and the proof for Lemma 4, all the agents in $R_i$ that does not reach $v_{min}^i$ stay at the same node $v_i'$. From such a situation, $na_i - (2g+1)$ agents in total try to move vertically to visit a node in a row ring with less than $2g+1$ agents (or node $v_{i',j}$ with $v_{i'j}.nAgentsAdjust[p] < 2g+1$). Intuitively, among the $na_i - (2g+1)$ agents, around half agents try to move up and another half agents try to move down. Concretely, let $a_h.rank$ be the ordinal number of how small its ID is among agents at the same node $v_{i,j}$ ($1 \leq a_h.rank \leq v_{ij}.nAgentsCurrent$). Then, when all the $na_i$ agents in $R_i$ stay at the same node $v_{min}^i$ (or $v_i'$), each agent $a_h$ with $1 \leq a_h.rank \leq \lfloor (na_i - (2g+1))/2 \rfloor$ (resp., $\lfloor (na_i-(2g+1))/2 \rfloor + 1 \leq a_h.rank \leq na_i-(2g+1)$) belongs to the *up group* (resp., *down group*), like Fig. 3(a). On the other hand, when there exist two nodes $v_{min}^i$ and $v_i'$ where an agent exists, let $v_{more}^i$ (resp., $v_{less}^i$) be the node where at least $\lceil na_i/2 \rceil$ (resp., at most $\lfloor na_i/2 \rfloor$) agents exist and let $na_{more}^i$ (resp., $na_{less}^i$) be the number of agents staying at $v_{more}^i$ (resp., $v_{less}^i$). Then, when $na_{more}^i \geq 2g+1$, each agent $a_h$ at $v_{more}^i$ with $1 \leq a_h.rank \leq \lfloor (na_{more}^i-(2g+1))/2 \rfloor$ (resp., $\lfloor (na_{more}^i-(2g+1))/2 \rfloor + 1 \leq a_h.rank \leq na_{more}^i-(2g+1)$) and each agent $a_{h'}$ at node $v_{less}^i$ with $1 \leq a_{h'}.rank \leq \lfloor na_{less}^i/2 \rfloor$ (resp., $\lfloor na_{less}^i/2 \rfloor + 1 \leq a_{h'}.rank \leq na_{less}^i$) belong to the up group (resp., the down group), like Fig. 3(b). When $na_{more}^i < 2g+1$, each agent $a_h$ at $v_{more}^i$ with $1 \leq a_h.rank \leq \lfloor (na_i - (2g+1))/2 \rfloor$ (resp., $\lfloor (na_i - (2g+1))/2 \rfloor + 1 \leq a_h.rank \leq na_i - (2g+1)$) belongs to the up group (reps., the down group), and agents at $v_{less}^i$ do not try to move in this classification, like Fig. 3(c).

Thereafter, for $n$ rounds, each agent in the up (resp., down) group tries to move up (resp., down) until it reaches a node in some row ring $R_{i'}$ with $na_{i'} < 2g+1$ (or no value of $na_{i'}$ is written). By this behavior, since each column ring is represented as a 1-interval connected ring, either an up group or a down group can visit a node in a row ring with less than $2g+1$ agents. By repeating such an adjusting phase and the previous counting phase $n$ times in total, we show that agents eventually reach a configuration such that there exist at least $2g+1$ agents in each of row rings (Lemma 5). Thus, they achieve $g$-partial gathering by applying the existing method [19] to each row ring independently.

The pseudocode of the adjusting phase is described in Algorithm 3. In Algorithm 3, each agent uses procedure *DecideDirection*() to determine the vertical direction it should move (or it should keep staying at the current node), whose pseudocode is described in Algorithm 4. For simplicity, we omit how to calculate $a_h.rank$ in Algorithm 4.

Concerning the adjusting phase, we have the following lemma.

▶ **Lemma 5.** *After executing the adjusting phase $n$ times in total, agents reach a configuration such that there exist at least $2g+1$ agents in each of row rings.*

**Proof.** First, we simply show that, when there exist at least two agents in $R_i$ at the beginning of the $p$-th adjusting phase, after $a_1^i$ and $a_2^i$ move right and left for $2n$ rounds (lines 1 to 15 of Procedure *Adjusting*()), the correct number $na_i$ of agents existing in a row ring $R_i$ is stored to variable $v_{ij}.nAgentsAdjust[p]$ of each node $v_{i,j}$'s whiteboard in $R_i$. By Procedure

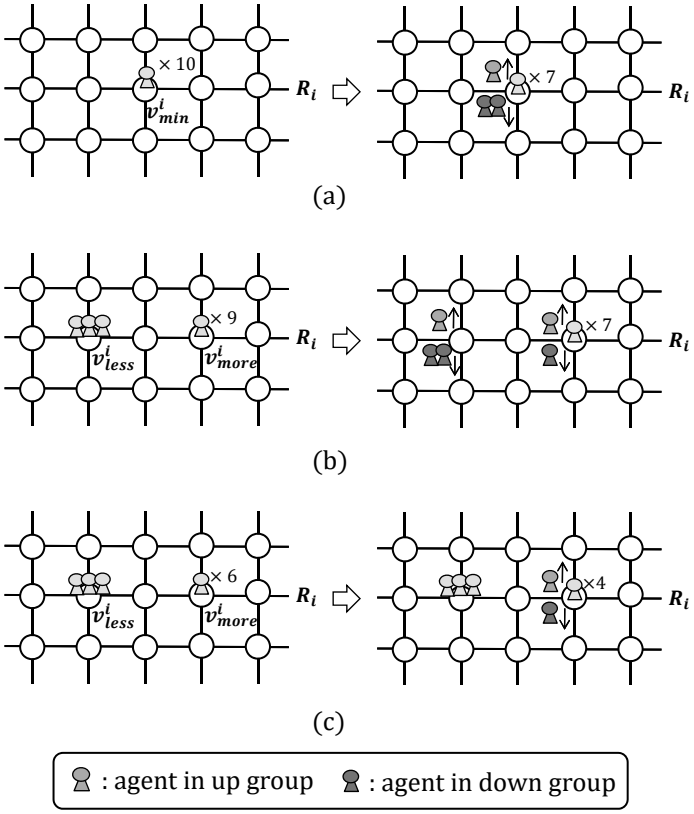



**Figure 3** Examples of forming up groups and down groups ($g = 3$).

*Adjusting*(), agent $a^i_1$ (resp., $a^i_2$) first tries to move right (resp., left) for $n$ rounds. When $a^i_1$ and $a^i_2$ start their behaviors at the same node, since at most one link is missing in each of row rings at each round, every node is visited within this $n$ rounds and the value of $na_i$ is stored to each node's whiteboard in $R_i$. When $a^i_1$ and $a^i_2$ start their behaviors at different nodes and there exists at least one unvisited node during the $n$ rounds, it means that $a^i_1$ and $a^i_2$ are blocked by the same link. In this case, for the next $n$ rounds, they switch their directions and $a^i_1$ (resp., $a^i_2$) tries to move left (resp., right). Then, by the similar discussion of the case when they start their movements at the same node, every node is visited wihtin this $n$ rounds and the value of $na_i$ is stored to each node's whiteboard in $R_i$. Thus, when there exist at least two agents in $R_i$ at the beginning of the $p$-th adjusting phase, the correct value of $na_i$ is stored to $v_{ij}.nAgentsAdjust[p]$ of each node $v_{i,j}$'s whiteboard in $R_i$.

In the following, we discuss the number of agents in each row ring after executing the adjusting phase $n$ times. In the proof, we assume that $k = 2gn + 2n - 1$ holds (we can show the lemma in the case of $k > 2gn + 2n - 1$ similarly). At the beginning of some adjusting phase, we call a row ring $R_i$ *enough* if there exist at least $2g + 1$ agents in $R_i$. Otherwise, we call the row ring *lacking*. In addition, we define the number $diff^i_{enough}$ (resp., $diff^i_{lack}$) of how more (resp., less) agents exist in row ring $R_i$ compared to $2g + 1$, and it is calculated as $na_i - (2g + 1)$ (resp., $(2g + 1) - na_i$). Notice that $diff^i_{enough}$ and $diff^i_{lack}$ are always non-negative numbers. Moreover, we define $SumDiff_{enough} = \sum_{i|R_i \text{ is enough}} diff^i_{enough}$ (resp., $SumDiff_{lack} = \sum_{i|R_i \text{ is lacking}} diff^i_{lack}$). Then, we first show the following claim.

**Algorithm 3** Procedure *Adjusting()* ($v_{i,j}$ is the current node of $a_h$).

**Main Routine of Agent $a_h$:**

```
1  a_h.rounds := 1
2  if a_h.nAgentsRowRing ≥ 2g + 3 then
3      if a_h.id is smallest among a_h.IDs[a_h.phase] then a_h.dir := 1
4      if a_h.id is the second smallest among a_h.IDs[a_h.phase] then a_h.dir := −1
5      while a_h.rounds ≤ n do
6          v_ij.nAgentsCurrent--
7          Try to move from the current node v_{i,j} to the neighboring node v_{i,j+a_h.dir}
8          if a_h reached v_{i,j+a_h.dir} (that becomes new v_{i,j}) then
               v_ij.nAgentsAdjust[a_h.phase] := a_h.nAgentsRowRing
9          v_ij.nAgentsCurrent++, a_h.rounds++
10     a_h.dir := a_h.dir × (−1)
11     while a_h.rounds ≤ 2n do
12         v_ij.nAgentsCurrent--
13         Try to move from the current node v_{i,j} to the neighboring node v_{i,j+a_h.dir}
14         if a_h reached v_{i,j+a_h.dir} (that becomes new v_{i,j}) then
               v_ij.nAgentsAdjust[a_h.phase] := a_h.nAgentsRowRing
15         v_ij.nAgentsCurrent++, a_h.rounds++
16     a_h.dir := 1
17     while a_h.rounds ≤ 5n do
18         if v_{i,j} is not the node where min{a_h.IDs[a_h.phase]} is not written in
               v_ij.IDs[v_ij.phase] then
19             v_ij.nAgentsCurrent--
20             Try to move from the current node v_{i,j} to the right neighboring node v_{i,j+1}
21             v_ij.nAgentsCurrent++
22         a_h.rounds++
23     a_h.dir := DecideDirection()
24     while a_h.rounds ≤ 6n do
25         if v_ij.nAgentsAdjust[a_h.phase] ≥ 2g + 1 then
26             v_ij.nAgentsCurrent--
27             Try to move from the current node v_{i,j} to the right neighboring node v_{i+a_h.dir,j}
28             v_ij.nAgentsCurrent++
29         a_h.rounds++
30 else
31     While a_h.rounds ≤ 6n do a_h.rounds++
32 Terminate the adjusting phase
33 // a_h increments its phase number and starts the next counting phase
```

▷ Claim 6. At the end of each adjusting phase, compared to the beginning of the adjusting phase, either of the following two properties holds: (i) The value of $SumDiff_{lack}$ at least halves, or (ii) The number of lacking rings decreases by at least one.

Proof. First, we briefly show that $SumDiff_{enough} = SumDiff_{lack} + n - 1$ always holds. Let $nr_{enough}$ (resp., $nr_{lack}$) be the number of enough (resp., lacking) row rings. Notice that $nr_{enough} = n - nr_{lack}$ holds. Then, there exist $nr_{lack} \times (2g+1) - SumDiff_{lack}$ agents in total in lacking row rings and hence there exist $2gn + 2n - 1 - (nr_{lack} \times (2g+1) - SumDiff_{lack})$ agents in total in enough row rings. Since the total number of agents in enough row rings can be also represented as $nr_{enough} \times (2g+1) + SumDiff_{enough}$, $2gn + 2n - 1 - (nr_{lack} \times (2g+1) - SumDiff_{lack}) = nr_{enough} \times (2g+1) + SumDiff_{enough}$ holds. Thus, since $nr_{enough} = n - nr_{lack}$ holds, the following equanality holds.

**Algorithm 4** Procedure *DecideDirection()* ($v_{i,j}$ is the current node of $a_h$).

**Main Routine of Agent $a_h$:**

```
1  a_h.dir := 0
2  if v_ij.nAgentsCurrent = a_h.nAgentsRowRing then
3      // All the agents in the current row ring stay at the same node
4      Calculate a_h.rank among the agents at the same node
5      if 1 ≤ a_h.rank ≤ ⌊(v_ij.nAgentsCurrent − (2g + 1))/2⌋ then a_h.dir := −1
6      else if
         ⌊(v_ij.nAgentsCurrent − (2g + 1))/2⌋ + 1 ≤ a_h.rank ≤ v_ij.nAgentsCurrent − (2g + 1)
         then a_h.dir := 1
7  else
8      // There exist two nodes where an agent exists in the current row ring
9      Calculate a_h.rank among the agents at the same node
10     if v_ij.nAgentsCurrent ≥ ⌈a_h.nAgentsRowRing/2⌉ then
11         if v_ij.nAgentsCurrent ≥ 2g + 1 then
12             if 1 ≤ a_h.rank ≤ ⌊(v_ij.nAgentsCurrent − (2g + 1))/2⌋ then a_h.dir := −1
13             else if
                 ⌊(v_ij.nAgentsCurrent−(2g+1))/2⌋+1 ≤ a_h.rank ≤ v_ij.nAgentsCurrent−(2g+1)
                 then a_h.dir := 1
14         else
15             if 1 ≤ a_h.rank ≤ ⌊(a_h.nAgentsRowRing − (2g + 1))/2⌋ then a_h.dir := −1
16             else if ⌊(a_h.nAgentsRowRing − (2g + 1))/2⌋ + 1 ≤ a_h.rank ≤
                 a_h.nAgentsRowRing − (2g + 1) then a_h.dir := 1
17     else
18         if a_h.nAgentsRowRing − v_ij.nAgentsCurrent ≥ 2g + 1 then
19             if 1 ≤ a_h.rank ≤ ⌊(v_ij.nAgentsCurrent)/2⌋ then a_h.dir := −1
20             else a_h.dir := 1
21 return a_h.dir
```

$$\begin{aligned} SumDiff_{enough} &= 2gn + 2n - 1 - (nr_{lack} \times (2g+1) - SumDiff_{lack}) - nr_{enough} \times (2g+1) \\ &= 2gn + 2n - 1 - (nr_{lack} \times (2g+1) - SumDiff_{lack}) - (n - nr_{lack}) \times (2g+1) \\ &= SumDiff_{lack} + n - 1. \end{aligned}$$

Then, by line 2 of Algorithm 3, since several agents in a row ring with at least $2g+3$ (resp., less than $2g+3$) agents try to (resp., do not try to) move vertically, the situation where the number of agents trying to move vertically from an enough row ring to a node in lacking row rings is the minimum, is that, among $nr_{enough}$ enough row rings, there exist $2g+2$ agents in each of $nr_{enough}-1$ enough row rings, and there exist the remaining $2g+1+SumDiff_{enough}-(nr_{enough}-1)$ agents in the other enough row ring. Then, by Algorithm 3 and the fact of $nr_{enough} \leq n$ and $SumDiff_{enough} = SumDiff_{lack} + n - 1$, the number of agents trying to move vertically is at least $2g+1+SumDiff_{enough}-(nr_{enough}-1)-(2g+1) = SumDiff_{enough}-(nr_{enough}-1) \geq SumDiff_{enough}-(n-1) = SumDiff_{lack}+n-1-(n-1) = SumDiff_{lack}$. In addition, since at most one link is missing in each of column rings, $\lfloor SumDiff_{lack}/2 \rfloor$ agents can visit a node in a lacking row ring $R^j_{lack}$. If $\lfloor SumDiff_{lack}/2 \rfloor \leq diff^j_{lack}$, $SumDiff_{lack}$ halves (property (i) holds). Otherwise, $R^j_{lack}$ becomes an enough row ring from the next adjusting phase (property (ii) holds). Therefore, the claim follows. ◁

Thus, by Claim 6 and the fact that the value of $SumDiff_{lack}$ is at most $= (2g+1)\times(n-1) = O(gn)$, after executing the adjusting phase $n$ times in total, $SumDiff_{lack}$ becomes 0, which means there exist at least $2g+1$ agents in each of row rings. Therefore, the lemma follows. ◀

We have the following theorem for the proposed algorithm.

▶ **Theorem 7.** *When $2gn+2n-1 \leq k \leq 2gn+6n+16g-12$, the proposed algorithm solves the $g$-partial gathering problem in dynamic tori with $O(n^2)$ rounds and the total number of $O(gn^3)$ moves.*

## 6 The case of $k \geq 2gn + 6n + 16g - 11$

In this section, when $k \geq 2gn+6n+16g-11$, we propose an algorithm to solve the problem with $O(n^2)$ rounds and the total number of $O(gn^2)$ (i.e., optimal) moves. In the algorithm, agents aim to make a configuration such that there exist at least $(n+8)g$ agents in some row ring $R_i$. Then, several agents that gathered in $R_i$ are partitioned into several groups each having at least $g$ agents and they try to visit all the nodes in the torus in total. During the movement, if some agent group starting from $R_i$ visits a node with less than $g$ agents, the less than $g$ agents join the group's movement. Thus, after all nodes are visited, agents achieve $g$-partial gathering.

Concretely, the algorithm comprises the following three phases: the observing phase, the semi-gathering phase, and the achievement phase. In the observing phase, each agent moves horizontally in the current row ring $R_i$ and recognizes the minimum agent ID among agents in $R_i$ (the actual behavior is almost the same as that of the counting phase in Section 5.1). In the semi-gathering phase, several agents in row ring $R_i$ move in the torus, observe the minimum ID written in each of row rings, and share the information with agents in $R_i$. Thereafter, each agent tries to move vertically to visit a node in the row ring $R_{min}$ where there exists an agent with the minimum ID among all agents in the initial configuration. However, there may exist agents that cannot reach a node in $R_{min}$ due to link-missings. Hence, in the achievement phase, agents in $R_{min}$ visit all the nodes in the torus in total to achieve $g$-partial gathering.

### 6.1 Observing phase

The behavior of agents in this phase is almost the same as that of the counting phase in Section 5.1. Concretely, each agent $a_h$ first writes its ID on the current node $v_{i,j}$'s whiteboard. Thereafter, for $3n$ rounds, $a_h$ tries to move right in its row ring $R_i$ and stores the observed IDs during the movement. By this behavior, by the similar discussion of the proof for Lemma 4, (i) each agent $a_h$ in row ring $R_i$ travels once around the row ring, or (ii) all the agents in $R_i$ stay at the same node. Then, in either case, each agent $a_h$ can get the list of IDs for all agents in $R_i$. Among the IDs, $a_h$ stores the minimum ID to variable $a_h.minIDrow$, and it selects the semi-gathering node $v_{sGather}$ as the node where the minimum ID is written in the row ring (the information of the semi-gathering node is used in the next subsection).

The pseudocode in the observing phase is described in Algorithm 5. Variables newly used from this section are given in Table 3. Notice that several variables are used in the following phases. Concerning the observing phase, by the same proof idea as that in Lemma 4, we have the following lemma.

▶ **Lemma 8.** *After finishing the observing phase, each agent $a_h$ in row ring $R_i$ stores the minimum agent ID among agents existing in $R_i$ in the initial configuration to variable $a_h.minIDrow$.*

**Table 3** Global variables newly used in Section 6.

**Variables for agent $a_h$.**

| Type | Name | Meaning | Initial value |
|---|---|---|---|
| int | $a_h.minIDrow$ | the minimum agent ID among agents existing in the same row ring as $a_h$ in $c_0$ | $\perp$ |
| int | $a_h.minIDall$ | the minimum agent ID among all the agents in $A_{row7}$ | $\perp$ |

**Variables for node $v_{i,j}$.**

| Type | Name | Meaning | Initial value |
|---|---|---|---|
| int | $v_{ij}.ID$ | ID stored at $v_{i,j}$ | $\perp$ |
| int | $v_{ij}.minIDrow$ | the minimum agent ID among agents existing in the row ring $R_i$ in $c_0$ | $\perp$ |
| int | $v_{ij}.minIDall$ | the minimum agent ID among all the agents in $A_{row7}$ | $\perp$ |

**Algorithm 5** The behavior of agent $a_h$ in the observing phase ($v_{i,j}$ is the current node of $a_h$).

**Main Routine of Agent $a_h$:**

1 $v_{ij}.ID := a_h.id, v_{ij}.nAgentsCurrent{+}{+}$
2 $a_h.nVisited := 1$, $a_h.rounds := 1$, $a_h.IDs[a_h.nAgentsRowRing] := a_h.id$
  $a_h.nAgentsRowRing{+}{+}$
3 **while** $a_h.rounds \leq 3n$ **do**
4  $v_{ij}.nAgentsCurrent$--
5  Try to move from the current node $v_{i,j}$ to the right neighboring node $v_{i,j+1}$
6  **if** $a_h$ *reached* $v_{i,j+1}$ *(that becomes new* $v_{i,j}$*)* **then**
7  $a_h.nVisited{+}{+}$
8  **if** $(v_{ij}.ID \neq \perp) \wedge (a_h.nVisited \leq n)$ **then**
9  $a_h.IDs[a_h.nAgentsRowRing] := v_{id}.ID$
10  $a_h.nAgentsRowRing{+}{+}$
11  $v_{ij}.nAgentsCurrent{+}{+}$, $a_h.rounds{+}{+}$
12 **if** $a_h.nVisited \geq n$ **then**
13  // $a_h$ traveled once around the row ring
14  Let $min$ be the minimum ID among $a_h.IDs[\,]$
15  $a_h.minIDrow := min$
16  Select the semi-gathering node $v_{sGather}$ as a node where $min$ (or $a_h.minIDrow$) is written
17 **if** $a_h.nVisited < n$ **then**
18  // all the agents in row ring $R_i$ stay at the current node
19  Let $min$ be the ID of agent $a_{h'}$ with $a_{h'}.rank = 1$
20  $a_h.minIDrow := min$
21  Select the current node as the semi-gathering node $v_{sGather}$
22 Terminate the counting phase and enter the semi-gathering phase

## 6.2 Semi-gathering phase

In this phase, agents try to move vertically to visit a node in the row ring $R$ so that there exist at least $(n+8)g$ agents in $R$. To this end, the semi-gathering phase comprises the following two sub-phases: the recognizing sub-phase and the moving sub-phase. In the recognizing sub-phase, several agents in each row ring $R_i$ move horizontally to write the value of the minimum agent ID among agents in $R_i$, move vertically to collect information of minimum IDs written in each row ring, and share the information of the row ring $R_{min}$ where, in the initial configuration, there exists the agent with the minimum ID among all agents that try to move in the torus in this sub-phase, with other agents in $R_i$. In the moving sub-phase, each agent moves vertically to visit a node in $R_{min}$.

### 6.2.1 Recognizing sub-phase

By Lemma 8, at the beginning of the recognizing sub-phase, each agent $a_h$ in row ring $R_i$ knows the list of IDs of all agents in $R_i$. When the number of IDs is less than 7 (i.e., there exist less than 7 agents in $R_i$ in the initial configuration), agents in $R_i$ do nothing in this sub-phase (and the next moving sub-phase) and wait for other agents' instructions, which is described in Section 6.3. Hence, in the following, we describe the behavior of agents in a row ring in which there exist at least 7 agents in the initial configuration. Notice that there exists at least one such a row ring with at least 7 agents because we assume $k \geq 2gn+6n+16g-11$ in this section and thus it never happens that there exist at most 6 agents in each row ring. We denote $R_{row7}$ by the set of such row rings and $A_{row7}$ by the set of agents existing in $R_{row7}$ at the beginning of the recognizing sub-phase. Let $a_1^i$ (resp., $a_2^i$) be the agent with the smallest (resp., the second smallest) ID among agents in $R_i$. Then, similar to the first behavior of the adjusting phase in Section 5.2 (Algorithm 3), $a_1^i$ (resp., $a_2^i$) tries to move right (resp., left) for $n$ rounds and then tries to move left (resp., right) for $n$ rounds. During the movement, $a_1^i$ and $a_2^i$ write the value of ID of $a_1^i$ ($= a_1^i.minIDrow$ or $a_2^i.minIDrow$) to variable $v_{ij}.minIDrow$ of each node $v_{i,j}$'s whiteboard. By this behavior, by the same proof idea of Lemma 5, the value of $a_1^i$'s ID is stored to each node's whiteboard in row ring $R_i$.

Thereafter, for $3n$ rounds, each agent $a_h$ tries to move right until visiting $v_{sGather}$ (where $a_1^i$ exists in the initial configuration). After the movement, by the similar discussion of the proof of Lemma 4, all the agents in $R_i$ that do not reach $v_{sGather}$ stay at the same node $v_i'$. Between $v_{sGather}$ and $v_i'$, we call the node with more (resp., less) agents $v_{more}^i$ (resp., $v_{less}^i$) (tie is broken using agent IDs), and let $na_{more}^i$ be the number of agents at $v_{more}^i$. Then, since there exist at least 7 agents in $R_i$, $na_{more}^i \geq 4$ holds and the four agents at $v_{more}^i$ try to move vertically to collect the information of minimum IDs written in each row ring. We use the procedure called *Splitting*(), introduced in [18]. Concretely, let $a_1^{iMore}$ (resp., $a_2^{iMore}$, $a_3^{iMore}$, and $a_4^{iMore}$) be the agent with the smallest (resp., the second, third, and fourth smallest) ID at $v_{more}^i$. We call $a_1^{iMore}$ and $a_2^{iMore}$ (resp., $a_3^{iMore}$ and $a_4^{iMore}$) the *up group* (resp., the *down group*). Then, each agent $a_h^{iMore}$ ($1 \leq h \leq 4$) tries to move up or down for $12n$ rounds. Concretely, for the first $3n$ rounds, $a_h^{iMore}$ tries to move up regardless of whether it belongs to the up group or the down group. Next, for the second $3n$ rounds, agents in the up (resp., down) group try to move up (resp., down). Thereafter, for the third $3n$ rounds, each agent $a_h^{iMore}$ tries to move up. Finally, for the last (or fourth) $3n$ rounds, agents in the up (resp., down) group try to move up (resp., down). During the movement, $a_h^{iMore}$ memorizes the value of the minimum ID that has ever observed to variable $a_h^{iMore}.minIDall$. By this behavior, we can show by [18] that either the up group or the down group travels ounce around the current column ring, which means the group can know the value of the minimum agent ID among $A_{row7}$. Without loss of generality, we assume that the up group ($a_1^{iMore}$ and $a_2^{iMore}$) traveled once around the column ring. Then, after *Splitting*(), for $n$ rounds, $a_1^{iMore}$ and $a_3^{iMore}$ (resp., $a_2^{iMore}$ and $a_4^{iMore}$) try to move up (resp., down) to visit a from where they started these movements (i.e., $v_{more}^i$). Since at most one link is missing in each of column rings, either $a_1^{iMore}$ or $a_2^{iMore}$ can visit $v_{more}^i$ and either $a_3^{iMore}$ or $a_4^{iMore}$ can also visit $v_{more}^i$. Hence, after the movement, there exist at least two agents at $v_{more}^i$, and they can know the minimum agent ID among $A_{row7}$. Among the two agents, let $a_{1'}^{iMore}$ (resp., $a_{2'}^{iMore}$) be the agent with a smaller (resp., larger) ID. Then, for $n$ rounds, $a_{1'}^{iMore}$ (resp., $a_{2'}^{iMore}$) tries to move right (resp., left) and writes the value of $a_h^{iMore}.minIDall$ ($h = 1', 2'$) to the variable $v_{ij}.minIDall$ for each node $v_{i,j}$'s whiteboard in $R_i$. Then, since $a_{1'}^{iMore}$ and $a_{2'}^{iMore}$ can visit all the $n$ nodes in $R_i$ in total, through $v_{ij}.minIDall$, each agent $a_h$ in $R_i$ can store the correct value of the minimum agent ID among $A_{row7}$ to $a_h.minIDall$.

The pseudocode of the recognizing sub-phase is described in Algorithm 6. In Algorithm 6, agents use Procedure *Splitting*(), whose pseudocode is described in Algorithm 7. For simplicity, in *Splitting*(), we omit the description of from which node $v_{sGather}$ or $v'$ agents try to move vertically using IDs in the case that there exist the same number of agents at both $v_{sGather}$ and $v'$. In addition, for simplicity, we omit the detailed description of whether the up group traveled once around the column ring or the down group did so.

Concerning the recognizing sub-phase, we have the following lemma.

▶ **Lemma 9.** *Let $A_{row7}$ be the set of agents such that there exist at least 7 agents in their initially belonging row ring in the initial configuration, and let $\mathcal{R}_{row7}$ be the set of row rings having agents in $A_{row7}$ in the initial configuration. Then, after finishing the recognizing sub-phase, each agent in $\mathcal{R}_{row7}$ recognizes the minimum agent ID among $A_{row7}$.*

### 6.2.2 Moving sub-phase

In this sub-phase, agents try to visit a node in the row ring $R_{min}$ where there exists the agent with the minimum ID among $A_{row7}$ in the initial configuration. First, for $3n$ rounds, each agent $a_h$ in row ring $R_i$ tries to move right until visiting a node $v^i_{sGather}$ where there exists an agent with the minimum ID among all the agents in the current row ring $R_i$ in the initial configuration. Then, by a similar discussion of the proof of Lemma 4, agents that do not reach $v^i_{sGather}$ stay at the same node $v'_i$. Next, $a_h$ calculates its rank among agents at the same node. If its rank is at most (resp., more than) the half of the number of agents at the current node, $a_h$ belongs to an up (resp., a down) group. Then, for $n$ rounds, until visiting a node in $R_{min}$, $a_h$ tries to move up (resp., down) if it is in an up (resp., a down) group. Since at most one link is missing in each column ring, either the up group or down group can visit a node in $R_{min}$ after the movement. By this behavior, we show in Lemma 10 that there exist at least $(n+8)g$ agents in $R_{min}$ after finishing the moving sub-phase.

The pseudocode of the moving sub-phase is described in Algorithm 8. Notice that, in the previous recognizing sub-phase, agents that belonged to an up group or a down group may stay at a node not in $\mathcal{R}_{row7}$ at the beginning of the moving sub-phase due to link-missings. In this case, such agents do nothing in this sub-phase and wait for other agents' instructions in the next phase (lines 2 and 3).

Concerning the moving sub-phase, we have the following lemma.

▶ **Lemma 10.** *After finishing the moving sub-phase, there exist at least $(n+8)g$ agents in some row ring.*

**Proof.** By the behavior for the moving sub-phase (Algorithm 8), all agents existing in $\mathcal{R}_{row7}$ try to visit a node in $R_{min}$ and at least the half of such agents can visit there. Then, the initial configuration such that the number of agents that try to visit a node in $R_{min}$ is the minimum is that (1) there exist 7 agents in $R_{min}$, (2) there exist 6 agents in each of $n-2$ row rings, and (3) there exist the remaining agents in one row ring $R_i$. In this case, each agent in the $n-2$ row rings does not move in this sub-phase. In addition, it is possible that two agents that, existed in $R_{min}$ in the initial configuration and belonged to an up or a down group in the previous recognizing sub-phase, do not stay at a node in $R_{min}$ or $R_i$ after the movement. The same thing holds for agents in $R_i$. Thus, at the beginning of the moving sub-phase, since there exist at least 5 agents in $R_{min}$ and they do not move in this sub-phase, and since there exist at least $k-7-6(n-2)-2 = k-6n+3$ agents in $R_i$, the number of agents in $R_{min}$ at the end of the moving sub-phase is at least

**Algorithm 6** The behavior of agent $a_h$ in the recognizing sub-phase ($v_{i,j}$ is the current node of $a_h$).

**Main Routine of Agent $a_h$:**

$a_h.rounds := 1$, $a_h.dir := 0$
**if** $a_h.nAgentsRowRing < 7$ **then**
  **while** $a_h.rounds < 23n$ **do** $a_h.rounds$++
  Terminate the semi-gathering phase and enter the achievement phase
**else**
  **if** $a_h.id$ is smallest among $a_h.IDs[]$ **then** $a_h.dir := 1$
  **if** $a_h.id$ is the second smallest among $a_h.IDs[]$ **then** $a_h.dir := -1$
  **while** $a_h.rounds \leq n$ **do**
    $v_{ij}.nAgentsCurrent$--
    Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i,j+a_h.dir}$
    **if** $a_h$ reached $v_{i,j+a_h.dir}$ (that becomes new $v_{i,j}$) **then** $v_{ij}.minIDrow := a_h.minIDrow$
    $v_{ij}.nAgentsCurrent$++, $a_h.rounds$++
  $a_h.dir := a_h.dir \times (-1)$
  **while** $a_h.rounds \leq 2n$ **do**
    $v_{ij}.nAgentsCurrent$--
    Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i,j+a_h.dir}$
    **if** $a_h$ reached $v_{i,j+a_h.dir}$ (that becomes new $v_{i,j}$) **then** $v_{ij}.minIDrow := a_h.minIDrow$
    $v_{ij}.nAgentsCurrent$++, $a_h.rounds$++
  **while** $a_h.rounds \leq 5n$ **do**
    **if** $v_{ij}.ID \neq v_{ij}.minIDrow$ **then**
      $v_{ij}.nAgentsCurrent$--
      Try to move from the current node $v_{i,j}$ to the right neighboring node $v_{i,j+1}$
      $v_{ij}.nAgentsCurrent$++
    $a_h.rounds$++
  $a_h.minIDall := a_h.minIDrow$
  *Splitting*()
  $a_h.rounds := 1$
  **while** $a_h.rounds \leq n$ **do**
    **if** $v_{ij}.minIDrow \neq a_h.minIDrow$ **then**
      $v_{ij}.nAgentsCurrent$--
      Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i+a_h.dir,j}$
      $v_{ij}.nAgentsCurrent$++
    $a_h.rounds$++
  **if** $v_{ij}.minIDrow = a_h.minIDrow$ **then**
    **if** $a_h$ visited all the nodes in the current column ring during *Splitting*() **then**
     $v_{ij}.minIDall := a_h.minIDall$
    $a_h.minIDall := v_{ij}.minIDall$
    **if** $a_h.rank = 1$ **then** $a_h.dir := 1$
    **else if** $a_h.rank = 2$ **then** $a_h.dir := -1$
    **while** $a_h.rounds \leq 2n$ **do**
      $v_{ij}.nAgentsCurrent$--
      Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i,j+a_h.dir}$
      $v_{ij}.minIDall := a_h.minIDall$, $v_{ij}.nAgentsCurrent$++, $a_h.rounds$++
  Terminate the recognizing sub-phase and enter the moving sub-phase

**Algorithm 7** Procedure *Splitting*() ($v_{i,j}$ is the current node of $a_h$.)

**Main Routine of Agent** $a_h$**:**
1 $a_h.rounds := 1$, $a_h.dir := 0$
2 **if** $(v_{ij}.nAgentsCurrent \geq \lceil a_h.nAgentsRowRing/2 \rceil) \wedge (v_{ij}.nAgentsCurrent \geq 4)$ **then**
3  **if** $1 \leq a_h.rank \leq 2$ **then** $a_h.dir := -1$
4  **else if** $3 \leq a_h.rank \leq 4$ **then** $a_h.dir := 1$
5 **while** $a_h.rounds \leq 3n$ **do**
6  $v_{ij}.nAgentsCurrent$--
7  Try to move from the current node $v_{i,j}$ to the up neighboring node $v_{i-1,j}$
8  $v_{ij}.nAgentsCurrent$++
9  **If** $v_{ij}.minIDrow < a_h.minIDall$ **then** $a_h.minIDall := v_{ij}.minIDrow$
10  $a_h.rounds$++
11 **while** $a_h.rounds \leq 6n$ **do**
12  $v_{ij}.nAgentsCurrent$--
13  Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i+a_h.dir,j}$
14  $v_{ij}.nAgentsCurrent$++
15  **If** $v_{ij}.minIDrow < a_h.minIDall$ **then** $a_h.minIDall := v_{ij}.minIDrow$
16  $a_h.rounds$++
17 **while** $a_h.rounds \leq 9n$ **do**
18  $v_{ij}.nAgentsCurrent$--
19  Try to move from the current node $v_{i,j}$ to the up neighboring node $v_{i-1,j}$
20  $v_{ij}.nAgentsCurrent$++
21  **If** $v_{ij}.minIDrow < a_h.minIDall$ **then** $a_h.minIDall := v_{ij}.minIDrow$
22  $a_h.rounds$++
23 **while** $a_h.rounds \leq 12n$ **do**
24  $v_{ij}.nAgentsCurrent$--
25  Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i+a_h.dir,j}$
26  $v_{ij}.nAgentsCurrent$++
27  **If** $v_{ij}.minIDrow < a_h.minIDall$ **then** $a_h.minIDall := v_{ij}.minIDrow$
28  $a_h.rounds$++
29 **if** $(a_h.rank = 1) \vee (a_h.rank = 3)$ **then** $a_h.dir := -1$
30 **else** $a_h.dir := 1$

$$\begin{aligned}(7-2) + \lfloor (k-6n+3)/2 \rfloor &\geq 5 + (k-6n+3)/2 - 1 \\ &\geq 4 + (2gn + 16g - 8)/2 \\ &= 4 + gn + 8g - 4 \\ &= (n+8)g\end{aligned}$$

The second inequality comes from the assumption of $k \geq 2gn + 6n + 16g - 11$. Therefore, the lemma follows. ◀

## 6.3 Achievement phase

In this phase, agents in $R_{min}$ move in the torus to achieve $g$-partial gathering. Intuitively, from $R_{min}$, some $2g$ agents visit a node in a row ring. Thereafter, the $2g$ agents visit all the nodes in the row ring in a way such that an agent group $A_r$ with some $g$ agents tries to move right and another agent group $A_l$ with the other $g$ agents tries to move left. In addition, during the movement, when $A_r$ or $A_l$ visits a node with less than $g$ agents, the less than $g$ agents join the group's movement. By executing such a behavior in each of the $n$ row rings, agents can achieve $g$-partial gathering.

**Algorithm 8** The behavior of agent $a_h$ in the moving sub-phase ($v_{i,j}$ is the current node of $a_h$.)

**Main Routine of Agent $a_h$:**
$a_h.rounds := 1$, $a_h.dir := 0$
**if** $v_{ij}.minIDall = \perp$ **then**
  **while** $a_h.rounds \leq 4n$ **do** $a_h.rounds$++
**else**
  **while** $a_h.rounds \leq 3n$ **do**
    **if** $v_{ij}.ID \neq v_{ij}.minIDrow$ **then**
      $v_{ij}.nAgentsCurrent$--
      Try to move from the current node $v_{i,j}$ to the right neighboring node $v_{i,j+1}$
      $v_{ij}.nAgentsCurrent$++,
    $a_h.rounds$++
  **if** $a_h.rank \leq \lceil v_{ij}.nAgentsCurrent/2 \rceil$ **then** $a_h.dir := -1$
  **else** $a_h.dir := 1$
  **while** $a_h.rounds \leq 4n$ **do**
    **if** $v_{ij}.minIDrow \neq a_h.minIDall$ **then**
      $v_{ij}.nAgentsCurrent$--
      Try to move from the current node $v_{i,j}$ to the neighboring node $v_{i+a_h.dir,j}$
      $v_{ij}.nAgentsCurrent$++,
    $a_h.rounds$++
Terminate the moving sub-phase and enter the achievement phase

To this end, each agent $a_h$ in $R_{min}$ first executes procedure *Counting*() in Section 5.1 for $3n$ rounds. Then, $a_h$ can count the number of agents currently existing in $R_{min}$. Thereafter, for $3n$ rounds, $a_h$ tries to move right until visiting the node $v_{i,j}^{min}$ with $v_{ij}^{min}.ID = a_h.minIDall$. Then, similarly to the previous discussion, all the agents in $R_{min}$ that do not reach $v_{i,j}^{min}$ stay at the same node $v'_{min}$. Between the two nodes, we call the node with more (resp., less) agents $v_{min}^{more}$ (resp., $v_{min}^{less}$) (tie is broken using agent IDs). In addition, let $R_{fromMin}^{\ell}$ ($0 \leq \ell \leq n-1$) be the $\ell$-th up row ring from $R_{min}$. Here, we say a row ring $R_i$ is $\ell$-th up from row ring $R_{i'}$ when $i' - i = \ell$ holds. Notice that $R_{fromMin}^{0}$ is $R_{min}$ itself. Then, agents execute the following sub-phases $n$ times: in the $\ell$-th sub-phase, $4g$ agents from $v_{min}^{more}$ or $v_{min}^{less}$ try to move vertically so that at least $2g$ agents visit a node in $R_{nomMin}^{\ell}$, and then at least $2g$ agents try to move horizontally to visit all the nodes in $R_{fromMin}^{\ell}$.

First, as a special case, we explain the behaviors of the 0-th sub-phase (i.e., movements in row ring $R_{fromMin}^{0}$ or $R_{min}$). Let $na_{min}^{more}$ (resp., $na_{min}^{less}$) be the number of agents staying at $v_{min}^{more}$. When $na_{min}^{less} \geq g$, agents in $R_{min}$ do nothing for $2n$ rounds. Otherwise, several agents at $v_{min}^{more}$ move horizontally to visit $v_{min}^{less}$. Concretely, each agent $a_h$ at $v_{min}^{more}$ with $1 \leq a_h.rank \leq g$ (resp., $g+1 \leq a_h.rank \leq 2g$) belongs to a *right-left group* $A_{rl}$ (resp., *left-right group* $A_{lr}$). Then, for $n$ rounds, each agent in $A_{rl}$ (resp., $A_{lr}$) tries to move right (resp., left). By this behavior, $A_{rl}$ and $A_{lr}$ can visit all the $n$ nodes in $R_{min}$ in total. During the movement, if $A_{rl}$ or $A_{lr}$ visits $v_{min}^{less}$, the agents that stayed at $v_{min}^{less}$ join the group's movement. Thereafter, for $n$ rounds, each agent in $A_{rl}$ (resp., $A_{lr}$) tries to move left (resp., right) until visiting $v_{min}^{more}$. By this behavior, $A_{rl}$ or $A_{lr}$ can return to $v_{min}^{more}$, there exist at most two nodes $v_{min}^{more}$ and $v_{min}^{less}$ with an agent, and both $v_{min}^{more}$ and $v_{min}^{less}$ have at least $g$ agents. Notice that the position of $v_{min}^{less}$ may change by these movements, but it does not affect the following explanations. In the following, we explain the behavior of each $i$-th sub-phase ($i \geq 1$) with the updated numbers of $na_{min}^{more}$ and $na_{min}^{less}$.

In the 1-st sub-phase, for $n$ rounds, each agent $a_h$ at $v^{more}_{min}$ with $1 \leq a_h.rank \leq 2g$ (resp., $2g+1 \leq a_h.rank \leq 4g$) belongs to an up (resp., a down) group and tries to move up (resp., down) until visiting a node $v^1_{fromMin}$ in $R^1_{fromMin}$. Since at most one link is missing in each column ring at each round, the up group or the down group can reach $v^1_{fromMin}$. Without loss of generality, we assume that the up group reached there. Thereafter, among agents in the up group, each agent $a_h$ with $1 \leq a_h.rank \leq g$ (resp., $g+1 \leq a_h.rank \leq 2g$) belongs to a right-left group $A_{rl}$ (resp., left-right group $A_{lr}$). Then, for $n$ rounds, each agent in $A_{rl}$ (resp., $A_{lr}$) tries to move right (resp., left). By this behavior, $A_{rl}$ and $A_{lr}$ can visit all the $n$ nodes in $R^1_{fromMin}$ in total. During the movement, when $A_{rl}$ or $A_{lr}$ visits a node with less than $g$ agents, the less than $g$ agents join the group's movement. However, if the number $na_{new}$ of agents in the updated group is at least $2g$, using IDs, only $g$ agents continue their movements and the remaining $na_{new} - g$ ($\geq g$) agents stay at the current node to reduce the total number of agent moves. Thereafter, for $n$ rounds, each agent in $A_{rl}$ (resp., $A_{le}$) tries to move left (resp., right) until returning to $v^1_{fromMin}$. By this behavior, $A_{rl}$ or $A_{lr}$ can reach $v^1_{fromMin}$.

Next, at the beginning of the 2-nd sub-phase, letting $C^{more}_{min}$ be the column ring including $v^{more}_{min}$, there exist at least $g$ agents at $v^1_{fromMin}$ (i.e., $A_{rl}$ or $A_{lr}$), $2g$ agents at some node in $C^{more}_{min}$ (i.e., the down group that may not have reached $v^1_{fromMin}$ in the 1-st sub-phase), and $na^{more}_{min} - 4g$ agents at $v^{more}_{min}$. From this situation, for $n$ rounds, until visiting a node $v^2_{fromMin}$ in $R^2_{fromMin}$, the $2g$ agents that may not have reached $v^1_{fromMin}$ in the 1-st sub-phase (the down group in this explanation) try to move down, the $g$ agents at $v^1_{fromMin}$ try to move up, and the $g$ agents at $v^{more}_{min}$ whose ID is either of the 1-st, 2nd, ..., or $g$-th smallest newly try to move up. By this behavior, at least $2g$ agents can reach $v^2_{fromMin}$ by the similar discussion of the 1-st sub-phase. Thereafter, the $2g$ agents try to move right or left and visit all the $n$ nodes in $R^2_{fromMin}$, and some less than $g$ agents at a node in $R^2_{fromMin}$ join a group's movement. Agents repeat such sub-phases until the number of agents at $v^{more}_{min}$ becomes less than $2g$ at the end of some sub-phase $\ell'$.

In the $(\ell'+1)$-th sub-phase, agents at $v^{less}_{min}$ execute the exact same behavior as that of agents at $v^{more}_{min}$ in the 1-st sub-phase. Thereafter, agents that existed at $v^{less}_{min}$ complete the remaining sub-phases (i.e., until agents that existed in $R_{min}$ execute the sub-phases $n$ times in total). Then, agents achieve $g$-partial gathering. Notice that it is possible no agent at $v^{less}_{min}$ moves in these sub-phases when there exist a large number of agents at $v^{more}_{min}$ at the beginning of the 1-st sub-phase. Agents at $v^{less}_{min}$ can determine whether or not they need to move in the torus and when they should start moving if they need to move, by using the information of $na^{less}_{min}$ and the total number of agents existing in $R_{min}$ calculated at the beginning of this achievement phase.

Concerning the achievement phase, we have the following lemma.

▶ **Lemma 11.** *After finishing the achievement phase, agents solve the $g$-partial gathering problem.*

**Proof.** First, in the 0-th sub-phase, when $na^{less}_{min} < g$, by the behavior for the achievement phase, $2g$ agents at $v^{more}_{min}$ are partitioned into a right-left group $A_{rl}$ with at least $g$ agents and a left-right group $A_{rl}$ with at least $g$ agents, and $A_{rl}$ (resp., $A_{rl}$) tries to move right (resp., left) for $n$ rounds and then tries to move left (resp., right) for $n$ rounds until returning to $v^{more}_{min}$. In addition, during the movement, when $A_{rl}$ or $A_{lr}$ visits node $v^{less}_{min}$ with less than $g$ agents, the less than $g$ agents join the group's movement. Hence, by this behavior, at least $g$ agents or no agent exists at each node in $R_{min}$. Thereafter, in the $\ell$-th sub-phase ($1 \leq \ell \leq n-1$), since at least $2g$ agents try to move up and another at least $2g$ agents try to move down to visit the $v_\ell$ in the $\ell$-th up ring $R^\ell_{fromMin}$ from $R_{min}$, at least $2g$ agents can

reach $v_\ell$. Then, by the same discussion in the case of the 0-th sub-phase, a right-left group $A_{rl}$ and a left-right group $A_{rl}$ visit all the nodes in $R^\ell_{fromMin}$ in total, at least $g$ agents or no agent exists at each node in $R^\ell_{fromMin}$, and $A_{rl}$ or $A_{lr}$ returns to $v_\ell$. Hence, by executing such a behavior in each row ring one by one, agents can achieve $g$-partial gathering.

In the following, we show that there exist the sufficient number of agents in $R_{min}$ to solve the problem at the beginning of the achievement phase. To execute the above behaviors in $r$ consecutive row rings, at least $(r+3)g$ agents are required ($4g$ agents are required when agents from $v^{more}_{min}$ or $v^{less}_{min}$ start their vertical movements for the first time, and additional $g$ agents are required otherwise). In addition, since agents start the above behavior from at most two nodes $v^{more}_{min}$ and $v^{less}_{min}$, the situation where the required number of agents staying in $R_{min}$ (i.e., agents staying at $v^{more}_{min}$ or $v^{less}_{min}$) is the maximum when agents starting from $v^{more}_{min}$ visit $r'$ ($\lceil n/2 \rceil \leq r' \leq n-1$) consecutive row rings in total is such that there exist $(r'+3)g+(g-1)$ agents at $v^{more}_{min}$ and there exist $(n-r'+3)g+(g-1)$ agents at $v^{less}_{min}$. Thus, $(r'+3)g+(g-1)+(n-r'+3)g+(g-1) < (n+8)g$ agents are required in $R_{min}$ at the beginning of the achievement phase to solve the problem. In Lemma 10, we already showed that such agents exist.

Therefore, the lemma follows. ◀

We have the following theorem for the proposed algorithm.

▶ **Theorem 12.** *When $k = O(gn)$ and $k \geq 2gn+6n+gn-11$, the proposed algorithm solves the $g$-partial gathering problem with $O(n^2)$ rounds and the total number of $O(gn^2)$ moves.*

## 7 Conclusion

In this paper, we considered the $g$-partial gathering problem of mobile agents in $n \times n$ dynamic tori and considered the solvability of the problem and the time and move complexity, focusing on the relationship between values of $k, n$, and $g$. First, we showed that agents cannot solve the problem when $k = o(gn)$, and showed that agents require a total number of $\Omega(gn^2)$ moves to solve the problem when $k = \Theta(gn)$. Second, we showed that the problem can be solved with $O(n^2)$ rounds and the total number of $O(gn^3)$ moves when $2gn+2n-1 \leq k \leq 2gn+6n+16g-12$. Finally, we showed that the problem can be solved with $O(n^2)$ rounds and the total number of $O(gn^2)$ moves when $k = O(gn)$ and $k \geq 2gn+6n+16g-11$. From these results, our algorithms can solve the partial gathering problem in dynamic tori with the asymptotically optimal number $\Theta(gn)$ of agents and the second algorithm is also asymptotically optimal in terms of the total number of agent moves.

Future works are as follows. First, we consider the lower bound on the time complexity to solve the problem. Next, we consider whether or not agents can solve the problem with the asymptotically optimal total number of agent moves when $k = \Omega(gn)$ but it is not in the range considered in Section 6. Finally, we will consider the solvability of the problem for agents with weaker capabilities. In this paper, as a first step to propose algorithms for solving the problem, we assumed that agents have distinct IDs, knowledge of $k$ and $n$, common sense of direction, and they behave fully synchronously. In addition, we assumed that each node has a whiteboard. However, at this stage, we do not know whether or not agents with weaker capability can also solve the problem (e.g., agents without distinct IDs, without the common sense of direction, or agents that behave semi-synchronously or asynchronously). Hence, we plan to consider algorithms using agents with such weak capabilities. We conjecture that, in any of the above cases, agents cannot solve the problem or require more total number of moves than the proposed algorithms.

## References


1 D. Baba, T. Izumi, F. Ooshita, H. Kakugawa, and T. Masuzawa. Linear time and space gathering of anonymous mobile agents in asynchronous trees. *Theoretical Computer Science*, 478:118–126, 2013.

2 S. Das, Di GA. Luna, D. Mazzei, and G. Prencipe. Compacting oblivious agents on dynamic rings. *PeerJ Computer Science*, 7:1–29, 2021.

3 Y. Dieudonné and A. Pelc. Anonymous meeting in networks. *Algorithmica*, 74(2):908–946, 2016.

4 Y. Dieudonné, A. Pelc, and V. Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing*, 44(3):844–867, 2015.

5 P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory. *DISC*, pages 242–256, 2008.

6 T. Gotoh, Y. Sudo, F. Ooshita, H. Kakugawa, and T. Masuzawa. Group exploration of dynamic tori. *ICDCS*, pages 775–785, 2018.

7 R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'agents: Applications and performance of a mobile-agent system. Softw., Pract. Exper., 32(6):543–573, 2002.

8 E. Kranakis, D. Krizanc, and E. Markou. Mobile agent rendezvous in a synchronous torus. *LATIN*, pages 653–664, 2006.

9 E. Kranakis, D. Krozanc, and E. Markou. The Mobile Agent Rendezvous Problem in the Ring. Synthesis Lectures on Distributed Computing Theory, Vol. 1, 2010.

10 D.B. Lange and M. Oshima. Seven good reasons for mobile agents. *CACM*, 42(3):88–89, 1999.

11 Di GA. Luna, S. Dobrev, P. Flocchini, and N. Santoro. Distributed exploration of dynamic rings. *Distributed Computing*, 33(1):41–67, 2020.

12 Di GA. Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, and G. Viglietta. Gathering in dynamic rings. *Theoretical Computer Science*, 811:79–98, 2018.

13 F. Ooshita, S. Kawai, H. Kakugawa, and T. Masuzawa. Randomized gathering of mobile agents in anonymous unidirectional ring networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1289–1296, 2014.

14 M. Shibata, S. Kawai, F. Ooshita, H. Kakugawa, and T. Masuzawa. Partial gathering of mobile agents in asynchronous unidirectional rings. *Theoretical Computer Science*, 617:1–11, 2016.

15 M. Shibata, N. Kawata, Y. Sudo, F. Ooshita, H. Kakugawa, and T. Masuzawa. Move-optimal partial gathering of mobile agents without identifiers or global knowledge in asynchronous unidirectional rings. *Theoretical Computer Science*, 822:92–109, 2020.

16 M. Shibata, D. Nakamura, F. Ooshita, H. Kakugawa, and T. Masuzawa. Partial gathering of mobile agents in arbitrary networks. *IEICE Transactions on Information and Systems*, 102(3):444–453, 2019.

17 M. Shibata, F. Ooshita, H. Kakugawa, and T. Masuzawa. Move-optimal partial gathering of mobile agents in asynchronous trees. *Theoretical Computer Science*, 705:9–30, 2018.

18 M. Shibata, Y. Sudo, J. Nakamura, and Y. Kim. Uniform deployment of mobile agents in dynamic rings. *SSS*, pages 248–263, 2020.

19 M. Shibata, Y. Sudo, J. Nakamura, and Y. Kim. Partial gathering of mobile agents in dynamic rings. *arXiv preprint* , 2022. `arXiv:2212.03457`.

20 M. Shibata and S. Tixeuil. Partial gathering of mobile robots from multiplicity-allowed configurations in rings. *SSS*, pages 264–279, 2020.

# Bond Percolation in Small-World Graphs with Power-Law Distribution

**Luca Becchetti**
Sapienza University of Rome, Italy

**Andrea Clementi**
University of Rome Tor Vergata, Italy

**Francesco Pasquale**
University of Rome Tor Vergata, Italy

**Luca Trevisan**
Bocconi University, Milan, Italy

**Isabella Ziccardi**
Bocconi University, Milan, Italy

## Abstract

*Full-bond percolation* with parameter $p$ is the process in which, given a graph, for every edge independently, we keep the edge with probability $p$ and delete it with probability $1 - p$. Bond percolation is studied in parallel computing and network science to understand the resilience of distributed systems to random link failure and the spread of information in networks through unreliable links. Moreover, the full-bond percolation is equivalent to the *Reed-Frost process*, a network version of *SIR* epidemic spreading.

We consider *one-dimensional power-law small-world graphs* with parameter $\alpha$ obtained as the union of a cycle with additional long-range random edges: each pair of nodes $\{u, v\}$ at distance $L$ on the cycle is connected by a long-range edge $\{u, v\}$, with probability proportional to $1/L^{\alpha}$. Our analysis determines three phases for the percolation subgraph $G_p$ of the small-world graph, depending on the value of $\alpha$.

- If $\alpha < 1$, there is a $p < 1$ such that, with high probability, there are $\Omega(n)$ nodes that are reachable in $G_p$ from one another in $\mathcal{O}(\log n)$ hops;
- If $1 < \alpha < 2$, there is a $p < 1$ such that, with high probability, there are $\Omega(n)$ nodes that are reachable in $G_p$ from one another in $\log^{\mathcal{O}(1)}(n)$ hops;
- If $\alpha > 2$, for every $p < 1$, with high probability all connected components of $G_p$ have size $\mathcal{O}(\log n)$.

**2012 ACM Subject Classification** Theory of computation → Random network models

**Keywords and phrases** Information spreading, gossiping, epidemics, fault-tolerance, network self-organization and formation, complex systems, social and transportation networks

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.3

**Related Version** *Full Version*: `https://arxiv.org/pdf/2205.08774.pdf`

**Funding** *Luca Trevisan and Isabella Ziccardi:* Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 834861).
*Luca Becchetti*: Supported by the ERC Advanced Grant 788893 AMDROMA, EC H2020RIA project SoBigData++ (871042), PNRR MUR project PE0000013-FAIR, PNRR MUR project IR0000013-SoBigData.it.
*Andrea Clementi*: Partially supported by Spoke 1 "FutureHPC & BigData" of the *Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC)* funded by *MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali" di R&S (M4C2-19) – Next Generation EU (NGEU).*

© Luca Becchetti, Andrea Clementi, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi;
licensed under Creative Commons License CC-BY 4.0



## 1 Introduction

Given a graph $G = (V, E)$ and a probability $p(e)$ associated to each edge $e \in E$, the *full-bond percolation* process[1] on $G$ is the construction of a random subgraph $G_p = (V, E_p)$ of $G$, called the percolation graph, obtained by selecting each edge $e \in E$ to belong to $E_p$ with probability $p(e)$, independently of the other edges. Often, the focus is on the *homogeneous* case in which all probabilities $p(e)$ are equal to the same parameter $p$. The main questions of interest in this case are, depending on the choice of $G$ and $p$, what is the typical size of the connected components of $G_p$ and the typical distances between reachable vertices.

The study of the percolation process originates from mathematical physics [14, 20, 23] and it has several applications in parallel and distributed computing and network science [1, 9, 13, 10, 12, 17]. For example, the study of network reliability in the presence of random link failures is equivalent to the study of the connectivity properties of the percolation graph of the network of links [12, 16].

The *independent cascade* is a process that models the spread of information and the influence of individual choices on others in social networks, and it is equivalent to a percolation process in a way that we explain next. In the independent cascade, we have a network $G = (V, E)$, an *influence* probability $p(e)$ associated to each edge, and a set $I_0 \subseteq V$ of network nodes that initially have a certain opinion[2]. The process evolves over time according to the following natural local rule: if a node $u$ acquired the opinion at time $t$, the node $v$ does not have the opinion, and the edge $(u, v)$ exists in $G$, then node $u$ will attempt to convince node $v$ of the opinion, and it will succeed with probability $p(u, v)$. All nodes that were successfully convinced by at least one of their neighbors at time $t$ will acquire the opinion, and will attempt to convince their neighbors at time $t + 1$ and so on. The independent cascade is studied in [13], where the problem of interest is to find the most "influential" initial set $I_0$. The resulting epidemic process is shown in [13] to be equivalent to percolation in the following sense: the distribution of nodes influenced by $I_0$ in the independent cascade process has the same distribution as the set of nodes reachable from $I_0$ in the percolation graph of $G$ derived using the probabilities $p(e)$.

The *Reed-Frost* process is one of the simplest and cleanest models of *Susceptible-Infectious-Recovered* (*SIR*) epidemic spreading on networks [22, 24]. This process is identical to independent cascade with a fixed probability $p(e) = p$ for all edges $e$. We can interpret nodes that acquired the opinion in the previous step as *Infectious* nodes that can spread the disease/opinion, nodes that do not have the disease/opinion as nodes *Susceptible* to the infection, and nodes that acquired the disease/opinion two or more steps in the past as *Recovered* nodes that do not spread the disease any more and are immune to it. The probability $p$ corresponds to the probability that a contact between an infectious person and a susceptible one causes a transmission of the disease from the former to the latter. The set $I_0$ is the initial set of infectious people at time 0. This process, being equivalent to independent cascade, is also equivalent to percolation [13]: the distribution of nodes that are infected and eventually recover in the Reed-Frost process is the same as the distribution of the set of nodes reachable from $I_0$ in the percolation graph of $G$ derived using the probabilities $p(e)$. Furthermore, the distribution of nodes that are infectious at time $t$ is precisely the distribution of nodes at distance $t$ from $I_0$ in the percolation graph (see [8]).

---

[1] We simply write bond percolation when no confusion arises.

[2] Or hold a certain piece of information, or perform a certain action, these are all equivalent views that lead to the same distributed process.

We are interested in studying full-bond percolation (and hence reliability under random link failure, independent cascade, and Reed-Frost epidemic spreading) in one-dimensional small-world graphs with power-law distribution of edges. *Small-world graphs* are a collection of generative models of graphs, designed to capture properties of real-life social networks [10, 22] and communication networks [11]. In the model introduced by Watts and Strogatz [25], the network is obtained as a one-dimensional or two-dimensional lattice in which certain edges are re-routed to random destinations. In a refined model introduced by Kleinberg [7, 10], a possible edge between two nodes at distance $L$ in the lattice exists with probability proportional to $L^{-\alpha}$, where the exponent $\alpha$ is a parameter of the model.

We study full bond percolation on the variant of Kleinberg's model defined below. This model has already been adopted in several previous papers [4, 5, 7, 10, 15, 22] to study bond percolation and epidemic processes, and to discover structural properties determining the performances of diffusion and navigations problems in real networks [26],

▶ **Definition 1** (1-D power-law small-world graphs). *For every $n \geqslant 3$ and $\alpha > 0$, an (undirected) random graph $G = (V, E)$ with $V = \{0, \ldots, n-1\}$ is sampled according to the distribution $\mathcal{SW}(n, \alpha)$ if $E = E_1 \cup E_2$, where: $(V, E_1)$ is a cycle and its edges are called* ring-edges*, and, for each pair of non-adjacent vertices $u, v \in V$, the* bridge *$\{u, v\}$ is included in $E_2$ independently, with probability*

$$\mathbf{Pr}\left(\{u, v\} \in E_2\right) = \frac{1}{d(u, v)^{\alpha}} \cdot \frac{1}{C(\alpha, n)},$$

*where $d(\cdot, \cdot)$ is the shortest-path distance in the ring and $C(\alpha, n)$ is the normalizing constant*[3] $C(\alpha, n) = 2\sum_{x=2}^{n/2} x^{-\alpha}$ .

The process of *long-range percolation* in the one-dimensional case is the variant of the percolation process applied to the generative model described above in which ring edges are preserved with probability one. The resulting percolation graphs are always connected, and the main question of interest is their diameter. Long-range percolation is well understood, and the one-dimensional case is studied in [4, 15, 25]. In particular, [4] provides bounds on the diameter and on the expansion of such graphs as a function of the power-law exponent $\alpha$. Such results have been then sharpened and generalized to multi-dimensional boxes in [6]. The long-range percolation process, however, is not a realistic generative model for epidemiological processes, because even the most contagious diseases, including Ebola, do not have 100% probability of spreading through close contacts (see [18, 21, 22] for a discussion of this point).

Full-bond percolation in power-law small-world graphs has been studied in the case of infinite lattices, including the one-dimensional case that is the infinite analog of the model that we study in this paper. In the infinite case, the main questions of interest, which are studied in [5], are whether the percolation graph has an infinite connected component and, given two vertices, what is their typical distance in the percolation conditioned on them both being in the infinite component, as a function of their distance in the lattice. Although there are similarities, techniques developed to study infinite percolation graphs do not immediately apply to the finite case.

[3] Note that $C(n, \alpha)$ is not, strictly speaking, a constant, but rather a normalizing factor that depends on both $\alpha$ and $n$. It is always upper bounded by an absolute constant across the entire range of $\alpha$, while it falls within an interval bounded by two constants when $\alpha > 1$. For the sake of conciseness, abusing terminology we write "constant" instead of normalizing factor.

## 1.1 Roadmap

In this paper, we study full-bond percolation in power-law small-world graphs. The rest of this paper is organized as follows. Section 2 provides an overview of our results and their main consequences. Section 3 introduces notation and some key preliminary notions, while Sections 4–6 provide the full analysis of full-bond percolation in one-dimensional power-law, small-world graphs. Each of Sections 4–6 includes a preliminary, informal description of the main techniques we adopt for that particular regime and a comparison with previous approaches.

Due to lack of space, the proofs of some technical results and a more extensive review of previous work is given in the full version of this paper [2].

# 2 Our Contribution

We analyze the bond percolation process over the small-world graph $G$ sampled according to the distribution $\mathcal{SW}(n, \alpha)$. Consistently with previous results in long-range percolation models [4], our analysis shows that the process exhibits three different behaviors determined by different values of $\alpha$. We formally state such results in the next three theorems.

▶ **Theorem 2** (Case $\alpha \in (2, +\infty)$)**.** *Let $\alpha > 2$ be a constant and $p < 1$ a percolation probability. Sample a graph $G = (V, E)$ from the $\mathcal{SW}(n, \alpha)$ distribution and let $G_p$ be the percolation graph of $G$ with percolation probability $p$. The following holds:*

1. *W.h.p., the connected components of $G_p$ have size at most $\mathcal{O}(\log n)$;*
2. *For each node $v \in V$ and for any sufficiently large $\ell$, with probability $1 - \Omega(\ell^{-(\alpha-2)/2})$, every node connected to $v$ in $G_p$ is at ring-distance no larger than $\mathcal{O}(\ell^2)$ from $v$.*

From an epidemiological point of view, this first regime is thus characterized by a negligible chance to observe an outbreak according to the Reed-Frost process, even in the presence of a large number (say some small root of $n$) of initially infected nodes (i.e. sources). In particular, the second claim of the theorem above strongly bounds the possible infected area of the ring graph.

The following case, determined by the range $1 < \alpha < 2$, shows the most interesting behavior.

▶ **Theorem 3** (Case $\alpha \in (1, 2)$)**.** *Let $\alpha \in (1, 2)$ be a constant and $p$ a percolation probability. Sample a graph $G = (V, E)$ from the $\mathcal{SW}(n, \alpha)$ distribution and let $G_p$ be the percolation graph of $G$ with percolation probability $p$. Then, constants $\underline{p}, \overline{p} \in (0, 1)$ (with $\underline{p} \leqslant \overline{p}$) exist such that the following holds:*

1. *If $p > \overline{p}$, w.h.p., there exists a set of $\Omega(n)$ nodes that induces a connected sub-component in $G_p$ with diameter $\mathcal{O}(\text{polylog}(n))$;*
2. *If $p < \underline{p}$, w.h.p. all the connected components of $G_p$ have size $\mathcal{O}(\log n)$.*

The first claim above implies that, if $p$ is sufficiently large (but still a constant smaller than 1), then, there is a good chance that few source nodes are able to infect a large (i.e. $\Omega(n)$) number of nodes and, importantly, this outbreak takes place at an almost exponential speed.

Finally, when $\alpha < 1$, we show the emergence of a behavior similar to that generated by one-dimensional small-world models with bridges selected according to the Erdős-Rényi distribution [3, 19].

▶ **Theorem 4** (Case $\alpha \in (0,1)$)**.** *Let $\alpha \in (0,1)$ and $p$ a percolation probability. Sample a graph $G = (V, E)$ from the $\mathcal{SW}(n, \alpha)$ distribution and let $G_p$ be the percolation graph of $G$ with percolation probability $p$. Then, constants $\overline{p}, \underline{p} \in (0,1)$ (with $\underline{p} \leqslant \overline{p}$) exist, such that the following holds:*

1. *If $p > \overline{p}$, w.h.p., there exists a set of $\Omega(n)$ nodes that induces a connected sub-component in $G_p$ with diameter $\mathcal{O}(\log n)$. Moreover, for a sufficiently large $\beta > 0$, for any subset $S \subseteq V$ of size $|S| \geqslant \beta \log n$, the subset of nodes within distance $\mathcal{O}(\log n)$ in $G_p$ from $S$ has size $\Omega(n)$, w.h.p.*
2. *If $p < \underline{p}$, w.h.p. all the connected components of $G_p$ have size $\mathcal{O}(\log n)$.*

In this last case, the presence of a sparse subset of relatively-long random bridges implies, above the probability threshold, that a few (i.e. $\Omega(\log n)$) sources w.h.p. generate a large outbreak at exponential speed.

We conclude this section by observing that three phases above are characterized by sharply different distributions of the typical length (measured according to the ring metric) of the bridges. To gauge the difference, consider the expectation, for a fixed vertex, of the sum of the lengths (in ring metric) of the bridges incident on the vertex, and call this expectation $BL_{\alpha,n}$.

When $\alpha < 1$, we have that $BL_{\alpha,n}$ is linear in $n$. When $1 < \alpha < 2$, then $BL_{\alpha,n}$ is of the form $O(n^{2-\alpha})$, going to infinity with $n$, but sublinearly in $n$. Finally, when $\alpha > 2$, $BL_{\alpha,n}$ is a constant that depends only on $\alpha$ and is independent of $n$. Nodes have, in expectation, only one bridge, so $BL_{\alpha,n}$ is an indication of how much we can advance on the ring by following one bridge.

When $\alpha < 1$, the bridges are basically as good as random edges, and we would expect a giant component to emerge even after full-bond percolation, if $p$ is large enough. When $\alpha > 2$, the bridges behave like a constant number of ring-edges, and we would not expect a large component when $p < 1$.

The case $1 < \alpha < 2$ is the one for which it is hardest to build intuition, and the fact that the bridges typically have length of the form $n^{1-\Omega(1)}$ might suggest that it would take $n^{\Omega(1)}$ steps to reach antipodal nodes. Previous work on long-range percolation had established a polylogarithmic diameter bound in the model in which ring-edges are not subject to percolation. In that model, all pairs of nodes are reachable in a polylogarithmic number of steps even though the typical bridge has length $n^{1-\Omega(1)}$, and this suggests that the shortest path structure is such that a small number of long bridges is used by several shortest paths. One thus would expect such a structure to be sensitive to full-bond percolation, and indeed the proof of [4] relies on the deterministic presence of the ring-edges. Instead, we prove that, when $1 < \alpha < 2$, w.h.p, most pairs of nodes are reachable from one another in a polylogarithmic number of hops after the full-bond percolation process.

## 3 Model and Preliminaries

In this paper, we study bond percolation of graphs sampled from $\mathcal{SW}(n, \alpha)$, as formalized in Definition 1. In this section, we define notation, key notions and tools that will be used throughout the rest of this paper. Further notation used in the proofs of specific results is introduced wherever it is used.

For the sake of completeness, we begin with the formal definition of bond percolation.

▶ **Definition 5** (Bond percolation)**.** *Given a graph $G = (V, E)$ and a real $p \in [0,1]$, its bond* percolation graph *$G_p$ is the random subgraph obtained from $G$ by removing each edge $e \in E$ independently, with probability $1 - p$.*

Considered any graph $G = (V, E)$ and $v \in V$, we denote by $\mathcal{N}_G(v)$ the neighborhood of $v$ in $G$, while $\deg_G(v) = |\mathcal{N}_G(v)|$ denotes the degree of $v$ in $G$. We omit the subscript when $G$ is clear from context.

If $G = (V, E_1 \cup E_2)$ is sampled from $\mathcal{SW}(n, \alpha)$ as in Definition 1, we say that a bridge $\{u, v\} \in E_2$ has *length* $d(u, v)$ (where $d(\cdot, \cdot)$ is as in Definition 1). We also say that $u$ is at *ring-distance* $d(u, v)$ from $v$ (and viceversa). Considered a bridge $\{u, v\}$, we say that $\{u, v\}$ is on the *clockwise* (respectively, *counter-clockwise*) side of $u$ if $d(u, v)$ corresponds to moving clockwise (respectively, counter-clockwise) along the cycle $(V, E_1)$ from $u$ to $v$.

Given a graph $G = (V, E)$ and $s \in V$, we denote by $\Gamma_G(s)$ the connected component containing $s$ in $G$. We may omit $G$ when clear from context, while with a slight abuse of notation, we simply write $\Gamma_p(s)$ for $\Gamma_{G_p}(s)$ when we refer to the percolation $G_p$ of some graph $G$, which will always be understood from context. Given $G = (V, E)$ and $S \subseteq V$, $\mathrm{diam}_G(S)$ is equal to the diameter of the subgraph of $G$ induced by $S$ if this is connected, otherwise $\mathrm{diam}_G(S) = \infty$. With a slight abuse of notation, we write $\mathrm{diam}_p(S)$ when $G$ is the percolation graph $G_p$. Given $G = (V, E)$ sampled from $\mathcal{SW}(n, \alpha)$ and any subgraph $H = (V, E')$ such that $E' \subseteq E$ (e.g., $G_p$), we associate a *ring-metric* to $H$, so that the *ring-distance* between $u, v$ is simply $d(u, v)$ defined above on $G$.

In the sections that follow, unless stated otherwise, probabilities are always taken over both the randomness in the sampling of $G$ from $\mathcal{SW}(n, \alpha)$ and over the randomness of the percolation. We further remark that our choice of the normalizing constant $C(\alpha, n)$ in Definition 1 entails $\mathbf{E}[\deg(v)] = 3$, while the following, preliminary fact follows from a straightforward application of Chernoff bound:

▷ Claim 6. Sample a graph $G = (V, E)$ from $\mathcal{SW}(n, \alpha)$. Then,

$$\mathbf{Pr}\left(\max_v \deg(v) \leqslant 4 \log n + 2\right) \geqslant 1 - \frac{1}{n}\,. \tag{1}$$

## 3.1 Galton-Watson Branching Processes

Our analysis of the percolation process in part relies on a reduction to the analysis of appropriately defined branching processes.

▶ **Definition 7** (Galton-Watson Branching Process). *Let $W$ be a non-negative integer random variable, and let $\{W_{t,i}\}_{t \geqslant 1, i \geqslant 1}$ be an infinite sequence of independent identically distributed copies of $W$. The* Galton-Watson branching process *generated by the random variable $W$ is the process $\{X_t\}_{t \geqslant 0}$ defined by $X_0 = 1$ and by the recursion*

$$X_t = \sum_{i=1}^{X_{t-1}} W_{t,i}\,.$$

*All properties of the process $\{X_t\}_{t \geqslant 0}$ (in particular, population size and extinction probability) are captured by the equivalent process $\{B_t\}_{t \geqslant 0}$, recursively defined as follows:*

$$B_t = \begin{cases} 1, & t = 0; \\ B_{t-1} + W_t - 1, & t > 0 \text{ and } B_{t-1} > 0; \\ 0, & t > 0 \text{ and } B_{t-1} = 0\,, \end{cases}$$

*where $W_1, W_2, \ldots$ are an infinite sequence of independent and identically distributed copies of $W$.*

In the remainder, when we refer to the Galton-Watson process generated by $W$, we always mean the process $\{B_t\}_{t \geqslant 0}$. In particular, we define $\sigma = \min\{t > 0 : B_t = 0\}$ (we set $\sigma = +\infty$ if no such $t$ exists). Note that, for any $T < \sigma$, we have $B_T = \sum_{t=1}^{T} W_t - T$.

## 4 The case $\alpha > 2$

We recall that $G_p$ is the percolation graph of $G$, sampled from the $\mathcal{SW}(n, \alpha)$ distribution. When $\alpha > 2$, we show that each component of the percolation graph $G_p$ has w.h.p. at most $\mathcal{O}(\log n)$ nodes. To prove this fact, we need to cope with the complex "connectivity" of $G_p$ yielded by the percolation of both ring-edges and the random bridges. To better analyze this structure, we introduce the notion of *$\ell$-graph* $G_p^{(\ell)}$ of $G_p$, where $\ell$ is any fixed integer $\ell > 0$. This new graph $G_p^{(\ell)}$ is in turn a one-dimensional small-world graph of $n/\ell$ "supernodes". It is defined by any fixed partition of $V$ of disjoint ring intervals, each of them formed by $\ell$ nodes: such $n/\ell$ intervals are the nodes, called *super-nodes*, of $G_p^{(\ell)}$. The set of edges of $G_p^{(\ell)}$ is formed by two types of random links: the *super-edges* that connect two adjacent super-nodes and the *super-bridges* connecting two non adjacent super-nodes of the $n/\ell$-size ring (for the formal definition of $G_p^{(\ell)}$ see Definition 10). We then prove that, for any $\alpha > 2$ and any $p < 1$, each super-node of this graph has:

1. Constant probability to have no neighbors (for every value of $\ell$);
2. Probability $\mathcal{O}(1/\ell^{\alpha-2})$ to be incident to a super-bridge.

We then design an appropriate BFS visit of $G_p^{(\ell)}$ (see Algorithm 1) that keeps the role of super-edges and super-bridges well-separated. In more detail, starting from a single super-node s, we show that the number of super-nodes explored at each iteration of the visit turns out to be dominated by a branching process having two distinct additive contributions: one generated by the new, visited percolated super-edges and the other generated by the new, visited percolated super-bridges. Special care is required to avoid too-rough redundancy in counting possible overlapping contributions from such two experiments. Then, thanks to Claims 1 and 2 above, we can prove that, for $\ell = \mathcal{O}(1)$ sufficiently large, w.h.p. this branching process ends after $\mathcal{O}(\log n)$ steps, and this proves that the connected component of s in $G_p^{(\ell)}$ has $\mathcal{O}(\log n)$ super-nodes. We also remark that our approach above also implies the following interesting result: with probability $1 - \mathcal{O}(1/\ell^{\alpha-2})$, all super-nodes connected to $v$ are at ring distance at most $\ell$ from $v$. This implies that, for each node $s$, the nodes in the connected component of $s$ in $G_p$ are within ring distance $\mathcal{O}(\ell^2)$ from $s$, with the same probability.

We now proceed formally with the proof of Theorem 2, which is a consequence of the two lemmas below.

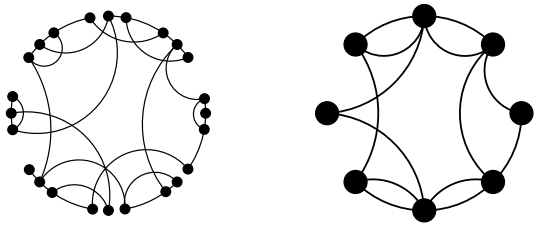
▶ **Lemma 8.** *Let $\alpha < 2$ be a constant and $p < 1$ a percolation probability. Sample a graph $G = (V, E)$ from the $\mathcal{SW}(n, \alpha)$ distribution and let $G_p$ its percolation graph. For every $s \in V = \{0, \ldots, n-1\}$, the connected component $\Gamma_p(s)$ contains $\mathcal{O}(\log n)$ nodes with probability at least $1 - 1/n^2$.*

▶ **Lemma 9.** *Under the same hypotheses of Lemma 8, for every $s \in V = \{0, \ldots, n-1\}$ and for any sufficiently large $\ell$,*

$$\mathbf{Pr}\left(\forall u \in \Gamma_p(s) : d(s,u) \leqslant 2\ell^2\right) \geqslant 1 - \frac{8}{(\alpha-2)\ell^{(\alpha-2)/2}}\,. \tag{2}$$

We remark that (2) implies that, for any increasing distance function $\ell = \ell(n) = \omega(1)$, every node in $\Gamma_p(s)$ has ring-distance from $s$ not exceeding $2\ell^2$ with probability $1 - o(1)$.

The proof of Lemma 8 is given in Section 4.2, while the proof of Lemma 9 in Section 4.3. To prove these lemmas, we rely on the notion of $\ell$-graph, defined in the following paragraph, together with supplementary notation that will be used in the remainder of this section.



(a) (b)

**Figure 1** (a) A graph $G = (V, E)$ with a ring-metric. (b) The 3-graph of $G$.

## 4.1 $\ell$-graphs

In what follows, we define a new graph on the vertex set $V = \{0, \dots, n-1\}$, starting from any one-dimensional small-world graph $G = (V, E_1 \cup E_2)$, where $(V, E_1)$ is a cycle, which defines a ring-metric, and $E_2$ is an arbitrary subset of bridges. In the remainder, we always assume $n \geqslant 3$.

▶ **Definition 10** ($\ell$-graph). *Let $H = (V, E_H)$ be a subgraph of a one-dimensional small-world graph $G$, where $E_H \subseteq E_1 \cup E_2$. For $\ell \geqslant 1$, consider an arbitrary partition of $V$ into $n/\ell$ disjoint intervals of length $\ell$, $\{I^{(1)}, \dots, I^{(n/\ell)}\}$ with respect to the ring metric induced by $(V, E_1)$. The* $\ell$-graph *associated to $H$ is the graph $H^{(\ell)} = (V^{(\ell)}, E_H^{(\ell)})$, where $V^{(\ell)} = \{I^{(1)}, \dots, I^{(n/\ell)}\}$, and*

$$E_H^{(\ell)} = \left\{ \{I^{(h)}, I^{(k)}\} : \exists u \in I^{(h)}, v \in I^{(k)} \text{ s.t. } \{u, v\} \in E_H \right\}.$$

A generic element in $V^{(\ell)}$ thus corresponds to $\ell$ consecutive nodes in the ring $(V, E_1)$, it is called *super-node*, and, depending on the context, is denoted by $\mathrm{v}$ or by its corresponding interval $I_{\mathrm{v}}$ in the partition $\{I^{(1)}, \dots, I^{(n/\ell)}\}$ of $V$. Fixed a partition, we denote with $E_1^{(\ell)}$ the set of links connecting two adjacent super-nodes in $V^{(\ell)}$ (that is, two adjacent intervals in $(V, E_1)$): notice that $(V^{(\ell)}, E_1^{(\ell)})$ is a ring of $n/\ell$ super-nodes, called *$\ell$-ring*. Then, given a subgraph $H = (V, E_H)$ as in the definition above, the elements in $E_H^{(\ell)}$ can be partitioned into two subsets: the elements in $E_H^{(\ell)} \cap E_1^{(\ell)}$ are called *super-edges*, while all the remaining elements in $E_H^{(\ell)}$ are called *super-bridges*. $H^{(\ell)}$ can thus be seen as a subgraph of a one-dimensional small-world graph $G^{(\ell)}$ with $n/\ell$ super-nodes, formed by a ring $(V^{(\ell)}, E_1^{(\ell)})$, and an additional set of super-bridges. The example in Figure 1 summarizes the above definitions. Let $H = (V, E_H)$ as in Definition 10 and assume $u, v \in V$, with $u \in I^{(h)}$ and $v \in I^{(k)}$. Clearly, if $(u, v) \in E_H$ then $(I^{(h)}, I^{(k)}) \in E_H^{(\ell)}$. Moreover, the following fact straightforwardly holds:

▷ Claim 11. Let $H = (V, E_H)$ as in Definition 10 and let $s \in I^{(k)}$ for some $k \in \{1, \dots, n/\ell\}$. Then,

$$|\Gamma_H(s)| \leqslant \ell |\Gamma_{H^{(\ell)}}(I^{(k)})|.$$

The following, preliminary lemma is used in the proofs of Lemma 8 and Lemma 9.

▶ **Lemma 12.** *Assume the hypotheses of Lemma 8 (and Lemma 9), and let ${G_p}^{(\ell)} = ({V}^{(\ell)}, {E_p}^{(\ell)})$ be the $\ell$-graph of $G_p$ generated by any fixed interval partition of $V$. Then, for each $\mathrm{v} \in V^{(\ell)}$, we have:*

$$\mathbf{Pr}\left(\deg_{{G_p}^{(\ell)}}(\mathrm{v}) = 0\right) \geqslant (1-p)^2 e^{-2/(\alpha-2)}, \tag{3}$$

$$\mathbf{Pr}\left(\mathrm{v} \text{ is incident to a super-bridge in } {G_p}^{(\ell)}\right) \leqslant \frac{2}{(\alpha-2)\ell^{\alpha-2}}. \tag{4}$$

**Proof.** Consider any super-node $\mathrm{v} \in V^{(\ell)}$ and, for the rest of this proof, denote by $w_1$ and $w_2$ the two boundary nodes of $I_{\mathrm{V}}$. Without loss of generality, we assume $I_{\mathrm{V}}$ includes the ring-edges that we traverse if we move on the cycle $(V, E_1)$ from $w_1$ to $w_2$ counter-clockwise.

We first prove (3). The super-node v has no out-edges in ${G_p}^{(\ell)}$ if and only if i) each node in $I_{\mathrm{V}}$ has no bridge to a node in $V \setminus I_{\mathrm{V}}$ and ii) $w_1$ and $w_2$ share no-ring edges with nodes in $V \setminus I_{\mathrm{V}}$ in $G_p$. Condition i) above is equivalent to the following: for every $x = 1, \ldots, \ell$, the node $u \in I_v$ at distance $d(w_1, u) = x$ from $w_1$ has no bridge of length exceeding $x$ on the clockwise side and of length exceeding $\ell - x$ on the counter-clockwise side. We have

$$\begin{aligned}
&\mathbf{Pr}\left(v \text{ has a bridge with length} > x \text{ in one side}\right) \leqslant \sum_{y=x+1}^{n/2} \frac{1}{C(\alpha,n)y^{\alpha}} \\
&\leqslant \int_{x+1}^{+\infty} \frac{1}{C(\alpha,n)(y-1)^{\alpha}} dy \leqslant \frac{1}{(\alpha-1)C(\alpha,n)x^{\alpha-1}} \leqslant \frac{1}{x^{\alpha-1}},
\end{aligned}$$

where the last inequality follows from the fact that $C(\alpha, n) \geqslant 1/2^{\alpha-1}$. From the inequality above, we have

$$\begin{aligned}
&\mathbf{Pr}\left(\deg_{{G_p}^{(\ell)}}(\mathrm{v}) = 0\right) \geqslant \left[(1-p) \cdot \prod_{x=1}^{\ell}\left(1 - \frac{1}{2x^{\alpha-1}}\right)\right]^2 \\
&\geqslant (1-p)^2 \cdot e^{-2\sum_{x=1}^{\ell} \frac{1}{x^{\alpha-1}}} \geqslant (1-p)^2 e^{-2/(\alpha-2)}.
\end{aligned}$$

We next prove (4). Let $v_i$ a node in $V \setminus I_{\mathrm{V}}$ at ring-distance $i + \ell$ from $I_{\mathrm{V}}$, i.e., such that $\min\{d(w_1, v_i), d(w_2, v_i)\} = i + \ell$. We have:

$$\begin{aligned}
&\mathbf{Pr}\left(v_i \text{ is not a neighbor of any node in } I_{\mathrm{V}} \text{ in } G_p\right) \\
&\geqslant \prod_{x=1}^{\ell}\left(1 - \frac{1}{C(\alpha,n)(x+\ell+i)^{\alpha}}\right) \geqslant e^{-\sum_{x=1}^{\ell} \frac{1}{(x+\ell+i)^{\alpha}}} \geqslant e^{-\frac{1}{(i+\ell)^{\alpha-1}}}.
\end{aligned}$$

Then, let ${(E_p)_2}^{(\ell)}$ denote the set of super-bridges in ${G_p}^{(\ell)}$, we use the above inequality to bound the expected number of super-bridges that are incident in v:

$$\begin{aligned}
&\mathbf{E}\left[\left|\mathcal{N}_{{G_p}^{(\ell)}}(\mathrm{v}) \cap {(E_p)_2}^{(\ell)}\right|\right] = 2\sum_{i=1}^{n/2-\ell} \mathbf{Pr}\left(v_i \text{ has a neighbor in } I_{\mathrm{V}}\right) \\
&\leqslant 2\sum_{i=1}^{+\infty} 1 - e^{-\frac{1}{(i+\ell)^{\alpha-1}}} \leqslant 2\sum_{i=1}^{+\infty} \frac{2}{(i+\ell)^{\alpha-1}} \leqslant \frac{2}{(\alpha-2)\ell^{\alpha-2}}.
\end{aligned}$$

Finally, the proof follows from

$$\mathbf{Pr}\left(\mathrm{v} \text{ has a super-bridge in } {G_p}^{(\ell)}\right) \leqslant \mathbf{E}\left[\left|\mathcal{N}_{{G_p}^{(\ell)}}(\mathrm{v}) \cap {(E_p)_2}^{(\ell)}\right|\right]. \qquad \blacktriangleleft$$

## 4.2 Proof of Lemma 8

To prove Lemma 8, we consider Algorithm 1 below that performs a BFS visit of the $\ell$-graph of the percolation graph $G_p$. Then, in Lemma 14, we prove that this algorithm terminates after visiting (only) $\mathcal{O}(\log n)$ super-nodes, w.h.p., for a sufficiently large $\ell = \mathcal{O}(1)$. Together with Claim 11, this proves Lemma 8.

**Algorithm 1** BFS visit of ${G_p}^{(\ell)}$.

1: **Input**: The $\ell$-graph ${G_p}^{(\ell)} = (V^{(\ell)}, {E_p}^{(\ell)})$ of $G_p$; an initiator (super-node) $\mathrm{S} \subseteq V^{(\ell)}$
2: $Q = \{\mathrm{S}\}$
3: **while** $Q \neq \emptyset$ **do**
4: $\quad \mathrm{W} = \texttt{dequeue}(Q)$
5: $\quad \texttt{visited}(\mathrm{W}) = \texttt{True}$
6: $\quad$ **for** each X s.t. $\{\mathrm{X}, \mathrm{W}\}$ is a super-edge of ${E_p}^{(\ell)}$ and $\texttt{visited}(x) = \texttt{False}$ **do**
7: $\quad\quad \texttt{enqueue}(\mathrm{X}, Q)$
8: $\quad$ **for** each Y s.t. $\{\mathrm{Y}, \mathrm{W}\}$ is a super-bridge of ${E_p}^{(\ell)}$ and $\texttt{visited}(\mathrm{Y}) = \texttt{False}$ **do**
9: $\quad\quad \texttt{enqueue}(\mathrm{Y}, Q)$
10: $\quad\quad \mathrm{Y}_{\text{left}}$ = the super-node at distance 1 from Y on the ring at its left
11: $\quad\quad$ **if** $\texttt{visited}(\mathrm{Y}_{\text{left}}) = \texttt{False}$ and $\{\mathrm{Y}_{\text{left}}, \mathrm{Y}\} \in {E_p}^{(\ell)}$ **then**
12: $\quad\quad\quad \texttt{enqueue}(\mathrm{Y}_{\text{left}}, Q)$

▶ Remark 13. Note that, each time we add a super-node Y to queue $Q$, we also also add the super-node $\mathrm{Y}_{\text{left}}$ to its left on the $\ell$-ring if $\mathrm{Y}_{\text{left}}$ is connected Y in $G_p^{(\ell)}$. So, in each while loop at line 3, for each super-node $\mathrm{W} \in Q$, at most one non-visited neighbor of W on the $\ell$-ring will be added at line 7 since one of them has been already added to $Q$ at the same while loop in which W has been added to $Q$ (see line 10).

▶ **Lemma 14.** *Assume the hypotheses of Lemma 8 and fix any node $s \in V$. For any fixed interval partition*[4] *of the vertex set $V$, consider the $\ell$-graph ${G_p}^{(\ell)}$ and the super-node $\mathrm{S} \in V^{(\ell)}$ such that $s \in I_S$. Then, a sufficiently large $\ell = \mathcal{O}(1)$ exists, depending only on $p$ and $\alpha$, such that Algorithm 1 terminates within $\mathcal{O}(\log n)$ iterations of the while loop in line 3, with probability at least $1 - 1/n^2$.*

**Proof.** For $t = 1, 2, \ldots$, let $Q_t$ be the content of $Q$ at the end of the $t$-th iteration of the while loop of Algorithm 1, and let $W_t$ denote the subset of super-nodes that were added to $Q$ during the $t$-th iteration. We have $|Q_0| = 1$ and

$$|Q_t| = \begin{cases} 0 \text{ if } |Q_{t-1}| = 0 \\ |Q_{t-1}| + |W_t| - 1 \text{ otherwise,} \end{cases}$$

Let $X_t$ and $Y_t$ denote the sets of super-nodes that were added to $Q$ in the $t$-th iteration of the while loop, respectively at line 7 and at line 9. For $t \geqslant 2$, whenever a super-node is added to the queue, the queue also contains a super-node at distance 1 from it on the $\ell$-ring, if the corresponding super-bridge is in $E_p^{(\ell)}$. We thus have the following formula $|W_t| = |X_t| + 2|Y_t|$, where $|X_t| \leqslant 1$ for $t \geqslant 2$ and $|X_1| \leqslant 2$. Let $\delta = 1 - p$, $\epsilon = 2 - \alpha$ and note that, from (3) and (4),

$$\mathbf{Pr}\left(|X_t| = 1\right) \leqslant 1 - \delta^2 e^{-2/\epsilon} \quad \text{and} \quad \mathbf{E}\left[|Y_t|\right] \leqslant \frac{2}{\epsilon \ell^{\epsilon}}. \tag{5}$$

Moreover, note that for every $t$, $|Q_t| = 0$, whenever $\sum_{i=1}^{t} |W_i| \leqslant t$. Then, we can write

$$\sum_{i=1}^{t} |W_i| - t = \sum_{i=1}^{t} |X_i| + 2 \sum_{i=1}^{t} |Y_i| - t\,,$$

[4] According to Definition 10.

where $|X_i|$ are $\{0,1\}$ random variables and, if w is the node extracted from $Q$ in the $i$-th iteration of the while loop, $|Y_i|$ is the number of super-bridge neighbors of w, so that it can also be written as the sum of random variables in $\{0,1\}$. Note that $\{X_t\}_t$ and $\{Y_t\}_t$ are not independent random variables, because of the conditions that appear in lines 6, 8 and 11 of Algorithm 1. Still, it is easy to show that $X_t$'s and $Y_t$'s are dominated by independent copies of two random variables $X$ and $Y$, such that $\mathbf{Pr}(|X|=1) = 1-\delta^2 e^{-2/\epsilon}$ and $\mathbf{E}[|Y|] \leqslant 1/(\epsilon\ell^\epsilon)$. Hence, Chernoff bound and (5) imply that, for a certain $t = \mathcal{O}(\log n)$ depending on $\delta$ and $\epsilon$,

$$\mathbf{Pr}\left(\sum_{i=1}^{t}|X_i| \geqslant \left(1-\frac{\delta^2}{2}e^{-2/\epsilon}\right)t\right) \leqslant \frac{1}{2n^2}, \quad \text{and} \quad \mathbf{Pr}\left(\sum_{i=1}^{t}|Y_i| \geqslant \frac{2t}{\epsilon\ell^\epsilon}\right) \leqslant \frac{1}{2n^2}.$$

As a result, with probability at least $1-1/n^2$, we have:

$$\sum_{i=1}^{t}|X_i| + 2\sum_{i=1}^{t}|Y_i| - t \leqslant t\left(\frac{4}{\epsilon\ell^\epsilon} - \frac{\delta^2}{2}e^{-2/\epsilon}\right).$$

Hence, we can choose $\ell = \mathcal{O}(1)$ (depending only on $\epsilon$ and $\delta$) large enough, so that with probability at least $1-1/n^2$, $\sum_{i=1}^{t}|W_i| < t$, so that $|Q_t| = 0$ for $t = \mathcal{O}(\log n)$. ◀

## 4.3 Proof of Lemma 9

Lemma 9 follows from Claim 11 and Lemma 15 below.

▶ **Lemma 15.** *Assume the hypotheses of Lemma 9. For sufficiently large $\ell$ depending only on $p$ and $\alpha$, consider the $\ell$-graph ${G_p}^{(\ell)}$ and the super-node $s \in V^{(\ell)}$ such that $s \in I_s$. Then, with probability at least*

$$1 - \tfrac{8}{(\alpha-2)} \cdot \ell^{-(\alpha-2)/2},$$

*all nodes in $\Gamma_{{G_p}^{(\ell)}}(s)$ are within ring-distance $\ell$ from s in the ring $(V^{(\ell)}, {E_1}^{(\ell)})$.*

**Proof.** Without loss of generality, we assume s $= 0$ and let $V^{(\ell)} = \{0,\ldots,n/\ell\}$. Next, we consider nodes on the ring at increasing distance from s $= 0$ moving counter-clockwise, proving that nodes at distance exceeding $\ell^{(2-\alpha)/2}$ are not part of $\Gamma_{{G_p}^{(\ell)}}(\mathrm{s})$, the connected component of s. To this purpose, denote by $K$ be the random variable indicating the ring-distance of the super-node v closest to s, such that v has no incident super-edges in ${G_p}^{(\ell)}$. From (3), and setting $\epsilon = 2-\alpha$ and $\delta = 1-p$, we have

$$\mathbf{Pr}\left(\text{v has no super-edges in } {G_p}^{(\ell)}\right) \geqslant \delta^2 e^{-2/\epsilon},$$

So, for $k = \ell^{\epsilon/2}$ and for $\ell$ large enough

$$\mathbf{Pr}(K > k) \leqslant (1-\delta^2 e^{-2/\epsilon})^k \leqslant \frac{1}{\ell^{\epsilon/2}}.$$

Moreover, denote by $B_k$ be the event $B_k = \{\text{the } k \text{ nodes nearest to s have no super-bridges}\}$. From (4), from the independence of the edge percolation events and using a union bound, we have for the complementary event $B_k^C$ that $\mathbf{Pr}(B_k^C) \leqslant k \cdot 2(\epsilon\ell^\epsilon)^{-1}$. Iterating the same argument for nodes on the clockwise side of s, if $k = \ell^{\epsilon/2}$ we have

$$\begin{aligned}&\mathbf{Pr}\left(\text{there is a node at distance} \leqslant k \text{ from s in } \Gamma_p^{(\ell)}(\mathrm{s})\right)\\ &\leqslant 2\mathbf{Pr}\left(\{K>k\} \cup B_k^C\right) \leqslant \frac{4}{\ell^{\epsilon/2}} + \frac{4}{\epsilon\ell^{\epsilon/2}},\end{aligned}$$

which completes the proof. ◀

## 5 The case $1 < \alpha < 2$

The aim of this section is to prove Theorem 3.

When $1 < \alpha < 2$, we make an analysis, based on a suitable inductive interval partitioning of the initial ring. In this case, our technique is inspired by the partitioning method used in [20] and [4]. However, though the high-level idea is similar, our analysis needs to address technical challenges that require major adjustments. In particular, the analysis of [20] only addresses and relies upon properties of infinite one-dimensional lattices with additional long-range links, applying Kolmogorov 0-1 law to the tail event of percolation. On the other hand, [4] borrows from and partly extends [20] to the finite case, proving a result similar to ours for finite one-dimensional lattices, but with important differences both in the underlying percolation model and in the nature of the results they obtain. As for the first point, in their model, ring-edges are deterministically present, while only long-range edges (what we call bridges in this paper) are affected by percolation. In contrast, in our model, *all* edges are percolated to obtain a realization of $G_p$. Consequently, the graph is deterministically connected in [4], while connectivity is something that only occurs with some probability in our setting. In particular, we face a subtler challenge, since we need to show the existence of a connected component that both spans a constant fraction of the node *and* has polylogarithmic diameter[5]. Moreover, their result for the diameter of the percolated graph only holds with probability tending to 1 in the limit, as the number of nodes grows to infinity, while we are able to show high probability.

We next provide a technical overview of our approach, highlighting the main points of the proof in which our percolation model requires a major departure from the approach of [4].

Similarly to [4], we consider a diverging sequence $\{N_k\}_k$ and for each $k$ we consider in $G_p$ an interval of length $N_k$ consisting of adjacent nodes on the ring. Departing from [4], we then prove that each interval of size $N_k$ contains a constant fraction $(1-\varepsilon_k)N_k$ of nodes, that induces a connected component of diameter $D_k$, with probability $1-\delta_k$, considering only edges *internal* to the interval and where $\varepsilon_k$ and $\delta_k$ are suitable constants that only depend on $k$, $p$ and $\alpha$. To prove this fact, for each $k$, we consider an arbitrary interval $I_k$ of size $N_k$ and we proceed inductively as follows:

1. We divide $I_k$ in $N_k/N_{k-1}$ smaller intervals of size $N_{k-1}$;
2. We assume inductively that, with probability $1-\delta_{k-1}$, each of these intervals contains $(1-\varepsilon_{k-1})N_{k-1}$ nodes that induces a connected component of diameter $D_{k-1}$;
3. With concentration arguments, we prove that a constant fraction of the intervals in which we divide $I_k$ have the above property, with probability that increases with $k$ (we call these intervals *good*);
4. We consider all good intervals of the previous point, and we prove that all these intervals are connected to each other with probability increasing in $k$.

The above reasoning therefore implies, for some $\varepsilon_k$, the existence of a fraction $(1-\varepsilon_k)N_k$ of nodes in $I_k$, that with increasing probability in $k$, induces a connected component of diameter $D_k = 2D_k + 1$. More details about the sequences $N_k, \varepsilon_k$ and $\delta_k$ can be found in the full proof. As mentioned earlier, our proof needs to specifically address percolation of both ring-edges and bridges. This in particular, means tackling points (2) - (4) above, which is a major challenge not present in [4] and is taken care of in the technical Lemma 18 and in

[5] As an example, we might have a connected component including a constant fraction of the nodes and linear diameter, containing a smaller connected component, still spanning a constant fraction of the nodes, but of polylogarithmic diameter.

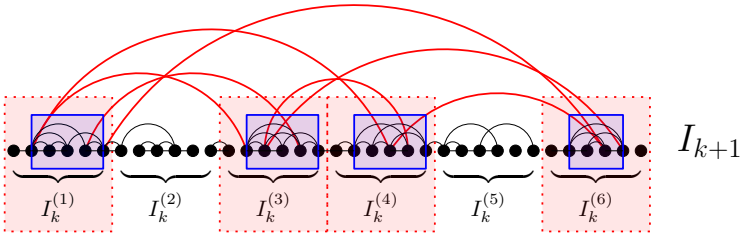



**Figure 2** Visualization of the proof of Lemma 16. Red dotted boxes identify "good" intervals in $H_{k+1}$ (intervals for which the event $A_k$ holds); for each good box, the included blue box contains the subset of nodes that induce a connected subgraph of size at least $(1-\varepsilon_k)N_k$ nodes and diameter $D_k$. In the lemma, we have to prove the existence of at least a certain number of red boxes, and that blue boxes are all mutually connected (i.e. the existence of the red edges in the picture).

the proof of the main Lemma 16 itself. In particular, considering only edges internal to the interval is very important, since, in this way, the events denoting the connection of disjoint intervals are independent. The recurrence of the diameter derives from the fact that the path connecting two nodes $u$ and $v$ in $I_k$ consists of the following three sub-paths:

1. The first part has length at most $D_k$, and it is a path in the aforementioned "good" interval of size $(1-\varepsilon_{k-1})N_{k-1}$ containing $u$;
2. The second part consists of a single edge, connecting the good interval containing $u$ with the good interval of $v$;
3. The third has length at most $D_k$, and is a path of the good interval of size $(1-\varepsilon_{k-1})N_{k-1}$ containing $v$.

Finally (Lemma 16), we consider $k$ such that $N_k = \Omega(n)$. For such $k$, we prove that

$$\delta_k = \mathcal{O}(1/n^{\varepsilon}), \quad \varepsilon_k \ll 1, \quad D_k = \text{polylog}(n),$$

so this implies the existence in $G_p$ of $\Omega(n)$ nodes inducing a connected component with diameter $\text{polylog}(n)$, w.h.p.

We now proceed with the formal analysis which requires the two lemmas below.

▶ **Lemma 16.** *Under the hypotheses of Theorem 3, assume $G = (V, E)$ is sampled from $\mathcal{SW}(n, \alpha)$ and let $G_p$ be its percolation graph. Then two constants $\overline{p} < 1$ and $\eta > 1$ exist such that, if $p > \overline{p}$, w.h.p. there is a set of $\Omega(n)$ nodes in $G_p$ that induces a connected sub-component with diameter $\mathcal{O}(\log^{\eta} n)$.*

▶ **Lemma 17.** *Under the hypotheses of Theorem 3, assume $G = (V, E)$ is sampled from $\mathcal{SW}(n, \alpha)$ and let $G_p$ its percolation graph. Then, a constant $\underline{p} > 0$ exists (in particular, $\underline{p} = 1/3$) such that, if $p < \underline{p}$, w.h.p. each connected component of $G_p$ has size at most $\mathcal{O}(\log n)$.*

The proof of Lemma 16 is given in Section 5.1, while the proof of Lemma 17 follows an approach, based on Galton-Watson processes, similar to that of Lemma 8, and it is given in Section 5.2.

## 5.1 Proof of Lemma 16

The lemma that follows is our new key ingredient to apply an inductive approach similar to the one in [4] in the case of full-bond percolation.

▶ **Lemma 18.** *Assume the hypotheses of Lemma 16, let* $\beta = \frac{1}{2}\alpha(3-\alpha)$ *and*

$$N_k = e^{\beta^k} \quad \text{and} \quad C_k = e^{\beta^{k-1}(\beta-1)}.$$

*Let* $I_k$ *be an arbitrary interval of* $N_k$ *adjacent nodes in the cycle* $(V, E_1)$*,* $D_k$ *any finite integer. Define the event* $A_k = \{\exists S \subseteq I_k : |S| \geqslant (1-\varepsilon_k)N_k \ \wedge \mathit{diam}_p(S) \leqslant D_k\}$*. Assume further that, for suitable, real constants* $\varepsilon_k, \delta_k$ *and* $p_k$ *in* $(0,1)$*,*

$$\mathbf{Pr}\,(A_k) \geqslant 1-\delta_k, \quad \text{if } p \geqslant p_k.$$

*Then, if we consider an interval* $I_{k+1}$ *of* $N_{k+1}$ *adjacent nodes, it holds:*

$$\mathbf{Pr}\,(A_{k+1}) \geqslant 1-\delta_{k+1}, \quad \text{if } p \geqslant p_{k+1},$$

*where*

$$\delta_{k+1} = 2C_{k+1}^{-0.2}, \quad \varepsilon_{k+1} = \varepsilon_k + \delta_k + C_{k+1}^{-0.2},$$
$$D_{k+1} = 2D_k + 1, \quad p_{k+1} = \frac{0.9(\alpha-1)(2-\alpha)}{(1-\varepsilon_k)^2(4.2-2\alpha)} C(n,\alpha).$$

**Proof.** In words, $A_k$ is the event that there exists a subset of the nodes in $I_k$ that induces a connected component of $G_p$ of size $\geqslant (1-\varepsilon_k)N_k$ nodes and diameter $\leqslant D_k$. Consider the interval $I_{k+1}$ and divide it into $C_{k+1}$ disjoint intervals of size $N_k$ (note that $N_{k+1} = N_k \cdot C_{k+1}$). Denote by $H_{k+1}$ the set of sub-intervals of $I_{k+1}$ of size $N_k$ for which event $A_k$ holds. In particular, for every $i = 1, 2, \ldots, C_{k+1}$, we use the indicator variable $A_k^{(i)}$ to specify whether or not $A_k$ holds for the $i$-th sub-interval, so that

$$|H_{k+1}| = \sum_{i=1}^{C_{k+1}} A_k^{(i)}.$$

where independence of the $A_k^{(i)}$'s follows from disjointness of the sub-intervals of $I_{k+1}$. For the sake of brevity, let $B_{k+1}$ denote the event $\{|H_{k+1}| \geqslant (1-\delta_k)C_{k+1} - C_{k+1}^{0.8}\}$. Application of Chernoff's bound then implies

$$\mathbf{Pr}\,(B_{k+1}) \geqslant 1 - e^{-2C_{k+1}^{0.6}} \geqslant 1 - C_{k+1}^{-0.2}.$$

Note that by definition, each sub-intervals in $H_{k+1}$ contains at least one subset (of the nodes) that induces a connected component in $G_p$ with at least $(1-\varepsilon_k)N_k$ nodes and diameter at most $D_k$. Our goal is to show that, with some probability, these intervals are all connected to each other: this in turn implies the existence of a set of nodes that induces a larger connected component, with diameter at most $2D_k+1$ and containing at least $|H_{k+1}|(1-\varepsilon_k)N_k$ nodes, see see Figure 2 for a visual intuition of the proof. In the remainder of this proof, we denote by $F_{k+1}$ the event that all connected components associated to intervals in $H_{k+1}$ are mutually connected. In particular, we prove that $\mathbf{Pr}\,(F_{k+1} \mid B_{k+1}) \geqslant 1 - C_{k+1}^{-0.2}$, so that

$$\mathbf{Pr}\,(A_{k+1}) = \mathbf{Pr}\,(F_{k+1} \cap B_{k+1}) = \mathbf{Pr}\,(B_{k+1})\,\mathbf{Pr}\,(F_{k+1} \mid B_{k+1}) \geqslant 1 - 2C_{k+1}^{-0.2},$$

which implies that $A_{k+1}$ holds with probability at least $1-\delta_{k+1}$, whenever we set

$$\delta_{k+1} = 2C_{k+1}^{-0.2} \quad \varepsilon_{k+1} = \varepsilon_k + \delta_k + C_{k+1}^{-0.2} \quad \text{and} \quad D_{k+1} = 2D_k + 1.$$

Now, we estimate the probability that, given $B_{k+1}$, $F_{k+1}$ holds. Two nodes in $I_{k+1}$ have distance at most $N_{k+1}$ on the cycle. Moreover, if we consider two sub-intervals of $I_{k+1}$ both belonging to $H_{k+1}$, the corresponding connected components contain at least $(1-\varepsilon_k)N_k$

nodes each, accounting for at least $(1-\varepsilon_k)^2 N_k^2$ pairs $\{u,v\}$, with $u$ belonging to the first and $v$ to second connected component. So, two given sub-intervals in $H_{k+1}$, they are not connected with probability at most

$$\left(1-\frac{p}{cN_{k+1}^{\alpha}}\right)^{(1-\varepsilon_k)^2 N_k^2} \leqslant \exp\left(-\frac{p}{c}(1-\varepsilon_k)^2\frac{e^{2\beta^k}}{e^{\alpha\beta^{k+1}}}\right) = \exp\left(-\frac{p}{c}(1-\varepsilon_k)^2 e^{\beta^k(2-\alpha\beta)}\right),$$

where in the remainder of this proof, we write $c$ for $C(\alpha,n)$, for the sake of readability. If we consider all pairs of intervals in $H_{k+1}$, a simple union bound allows us to conclude that the probability that the intervals in $H_{k+1}$ are not all mutually connected is at most

$$C_{k+1}^2 \exp\left(-\frac{p}{c}(1-\varepsilon_k)^2 e^{\beta^k(2-\alpha\beta)}\right) \leqslant \exp\left(\beta^k\left(2(\beta-1)-\frac{p}{c}(1-\varepsilon_k)^2(2(2-\alpha)+0.2)\right)\right),$$

where the last inequality follows from the definition of $\beta$. Finally, the quantity above can be upper bounded as follows

$$\exp\left(-\beta^k 0.1(\alpha-1)(2-\alpha)\right) = \exp(-0.2\beta^k(\beta-1)) = C_{k+1}^{-0.2},$$

whenever $p$ satisfies $p \geqslant \frac{c(2-\alpha)(\alpha-1)0.9}{(1-\varepsilon_k)^2(2(2-\alpha)+0.2)}$. ◀

Now we are ready to prove Lemma 16.

**Proof of Lemma 16.** Let $C_k$ and $N_k$ defined as in the claim of Lemma 18. First, we consider the series $\sum_k C_k^{-0.2}$ and, since $C_k = e^{\beta^{k-1}(\beta-1)}$ with $\beta > 1$, we notice that the series is convergent, i.e. $\sum_{k=1}^{\infty} C_k^{-0.2} < +\infty$.[6] This means that the tail of the series converges to zero, and this implies that there exists a constant $h$ such that

$$\sum_{k=h}^{+\infty} C_k^{-0.2} \leqslant \frac{1}{100}. \tag{6}$$

The constant $h$ depends only on $\alpha$. In particular, it increases as $\alpha \to 2^-$ and $\alpha \to 1^+$.

Now we consider $I_h$, an arbitrary interval of size $N_h = e^{\beta^h}$. Next, we let

$$\delta_h = 1 - p^{e^{\beta^h}}, \quad \varepsilon_h = 0, \quad D_h = e^{\beta^h}, \quad p_h = p,$$

and we consider the event $A_h$, defined as in the statement of Lemma 18. If no ring edge belonging to $I_h$ is percolated, $A_h$ is trivially true: this implies that, for every $p \geqslant p_h$, $\mathbf{Pr}(A_h) \geqslant 1-\delta_h$, where probability is over the edges with endpoints in $I_h$. If $A_{h+1}$ and $N_{h+1}$ are defined like in its statement, Lemma 18 then implies that, for an arbitrary interval $I_{h+1}$ of size $N_{h+1}$, we have

$$\mathbf{Pr}(A_{h+1}) \geqslant 1-\delta_{h+1},$$

if $p \geqslant p_{h+1}$ and whenever we take

$$\delta_{h+1} = 2C_{h+1}^{-0.2}, \quad \varepsilon_{h+1} = 1 - p^{e^{\beta^h}} + C_{h+1}^{-0.2},$$
$$D_{h+1} = 2e^{\beta^h} + 1, \quad p_{h+1} = \frac{0.9c(\alpha-1)(2-\alpha)}{(4.2-2\alpha)},$$

where the probability is taken over the randomness of the edges with endpoints in $I_{h+1}$.[7]

[6] We did not try to optimize constants and the choice 0.2 for the exponent is not necessarily optimal.
[7] Recall that $c = C(\alpha,n)$ in the remainder of this proof.

If we iteratively apply Lemma 18, we thus have for each $k \geqslant 1$ such that $N_k \leqslant n$,

$$\mathbf{Pr}(A_k) \geqslant 1 - \delta_k,$$

where $\delta_k = 2C_k^{-0.2}$, $p_k = \frac{0.9c(\alpha-1)(2-\alpha)}{(1-\varepsilon_{k-1})^2(4.2-2\alpha)}$, and $\varepsilon_k$ and $D_k$ are defined by the recurrencies:

$$\begin{cases} \varepsilon_k = \varepsilon_{k-1} + \delta_{k-1} + C_k^{-0.2}, \text{ if } k > h \\ \varepsilon_h = 1 - p^{e^{\beta^h}} \end{cases} \quad \text{and} \quad \begin{cases} D_k = 2D_{k-1} + 1, \text{ if } k > h \\ D_h = e^{\beta^h}. \end{cases}$$

Now, we solve the recurrence for $\varepsilon_k$, obtaining $\varepsilon_k = 1 - p^{e^{\beta^h}} + 3\sum_{i=h}^{k} C_i^{-0.2}$, where, leveraging (6), we take $p > \sigma$, with $\sigma > 0$ such that $1 - \sigma^{e^{\beta^h}} = 1/50$. With this choice, for each $k \geqslant 1$ we obtain

$$\varepsilon_k \leqslant 1 - p^{e^{\beta^h}} + \frac{3}{100} \leqslant \frac{1}{20}.$$

Moreover, for each $i \leqslant k$, we have that $D_k \leqslant 2^{k-h+1} e^{\beta^h}$. If we take $m = \log_\beta(\log n)$, then $N_m = e^{\beta^m} = n$ and $C_m = n^{(\beta-1)/\beta}$. We also have:

$$D_m \leqslant \frac{e^{\beta^h}}{2^{h-1}} 2^{\log_\beta(\log(n))} = \frac{e^{\beta^h}}{2^{h-1}} (\log n)^{\log_\beta 2}.$$

Setting $\eta = \log_\beta 2$, we have $\eta > 1$, since $\beta < 2$. We thus have $D_m = O((\log^\eta n))$. Moreover, if $p \geqslant \sigma$

$$\delta_m \leqslant 2n^{-0.2(\beta-1)/\beta}, \ \varepsilon_m \leqslant \frac{1}{20}, \ p_m \geqslant \frac{0.9c(\alpha-1)(2-\alpha)}{(19/20)^2(4.2-2\alpha)}.$$

Finally, if

$$p \geqslant \max\left\{\sigma, \frac{0.9c(\alpha-1)(2-\alpha)}{(19/20)^2(4.2-2\alpha)}\right\} := \overline{p}$$

then,

$$\mathbf{Pr}(A) \geqslant 1 - \delta_m \geqslant 1 - 2n^{-0.2(\beta-1)/\beta},$$

where

$$A = \left\{\exists S \subseteq V : |S| > \frac{19}{20} n \wedge \operatorname{diam}_p(S) = O(\log^\eta(n))\right\},$$

i.e., w.h.p. $G_p$ contains an induced subgraph of size at least $(19/20)n$ nodes and diameter $O(\log^\eta(n))$. ◀

## 5.2 Proof of Lemma 17

Let $\underline{p} = 1/3$ and consider an arbitrary node $s \in V$. We consider an execution of the BFS in Algorithm 2 with input the percolation subgraph $G_p = (V, E_p)$ of $G = (V, E)$ and the source $s$.

**Algorithm 2** BFS visit of $G_p$.

```
1: Input: the subgraph G_p = (V, E_p), a source s ∈ V
2: Q = {s}
3: R = ∅
4: while Q ≠ ∅ do
5:     w = dequeue(Q)
6:     R = R ∪ {w}
7:     for each neighbor x of w in G_p such that x ∉ R do
8:         enqueue(x, Q)
```

We consider the generic $t$-th iteration of the while loop at line 4 and we denote by $W_t$ the number of nodes added to the queue $Q$ at line 8. Moreover, $B_t$ is the set of nodes that are in $R$ in the $t$-th iteration. By its definition, $B_t$ is a branching process described by the following recursion:

$$\begin{cases} B_t = B_{t-1} + W_t - 1 & \text{if } B_{t-1} \neq 0 \\ B_t = 0 & \text{if } B_{t-1} = 0 \\ B_0 = 1. \end{cases} \tag{7}$$

Note that, from Definition 1, each node $w \in V$ has expected degree $\mathbf{E}[\deg(w)] = 3$. So, since each edge in $G$ is also in $G_p$ with probability $p$, we have $\mathbf{E}[W_1] = 3p$ and $\mathbf{E}[W_t] \leqslant 3p$ for $t > 1$.[8] Since $p < \underline{p}$, there is a constant $\delta$ such that $p = (1-\delta)/3$ and, for each $t \geqslant 0$

$$\mathbf{E}[W_t] = 1 - \delta.$$

We consider the $T$-th iteration of the while loop, where $T = \gamma \log n$. Note that the random variables $W_1, \ldots, W_T$ are not independent as noted earlier but, as remarked in the proof of Lemma 14, it is easy to show that they are stochastically dominated by $T$ independent random variables distributed as $W_1$. For the sake of simplicity, we abuse notation, by using $W_1, \ldots, W_T$ to denote the $T$ independent copies of $W_1$ in the remainder of this proof. Now, if

$$\sum_{i=1}^{T} W_i - T < 0,$$

then $B_T = 0$. We notice that each $W_i$ can be written as a sum of $n+2$ independent Bernoulli random variables.[9] Hence, applying Chernoff's bound to $W = \sum_{i=1}^{T} W_i$ we obtain:

$$\mathbf{Pr}\left(W > (1+\delta)\mathbf{E}[W]\right) \leqslant e^{-\frac{\delta^2}{2}\mathbf{E}[W]}.$$

Next, since $\mathbf{E}[W] < (1-\delta)T$,

$$\mathbf{Pr}\left(W > (1-\delta^2)T\right) \leqslant e^{-\frac{\delta^2(1-\delta)}{2}T} \leqslant \frac{1}{n^2},$$

where the last inequality follows if we take $\gamma \geqslant 4/(\delta^2(1-\delta))$. This allows us to conclude that

$$\mathbf{Pr}\left(B_T = 0\right) \geqslant 1 - \frac{1}{n^2},$$

[8] We have not strict equality for $t > 1$, which follows since the $W_t$'s are not independent in general, since line 8 is only executed if $x \notin R$.

[9] Assume node $v$ is visited in the $t$-iteration. Then we have 2 indicator variables for the 2 ring edges incident in $v$, plus $n$ indicator variables, corresponding to $n$ bridges potentially incident in $v$.

for some $T = \mathcal{O}(\log n)$, which in turn implies that with the above probability, the connected component of which $v$ is part contains at most $\mathcal{O}(\log n)$ nodes. Finally, a union bound over all nodes in $V$ concludes the proof.

## 6 The case $\alpha < 1$

In this section, we prove Theorem 4. When $\alpha < 1$, we first notice that every bridge-edges $(u, v)$ is in $G_p$ with probability at least $pc/n$, for some constant $c$. This reduces our problem to the analysis of the percolation graph $H_p$ of a graph $H$, where $H$ is the union of a ring and an Erdős-Rényi graph $\mathcal{G}_{n,q}$. [3] contains a detailed analysis of this case, and we use their results to prove that $G_p$, for any sufficiently large $p$, contains a connected component with diameter $\mathcal{O}(\log n)$.

We finally remark that, for every value $\alpha$, when $p < 1/3$ the connected components of $G_p$ have at most $\mathcal{O}(\log n)$ nodes. This fact easy follows from concentration techniques, and the fact that every node in $G_p$ has expected degree at most 3.

The formal proof of Theorem 4 requires the two lemmas below.

▶ **Lemma 19.** *Under the hypotheses of Theorem 4, sample a $G = (V, E)$ from the $\mathcal{SW}(n, \alpha)$ distribution and let $G_p$ be its percolation subgraph. Then, a constant $\overline{p} < 1$ exists such that, if $p > \overline{p}$ then, w.h.p., there is a set of $\Omega(n)$ nodes in $G_p$ that induce a connected subgraph of diameter $\mathcal{O}(\log n)$.*

▶ **Lemma 20.** *Under the hypothesis of Theorem 4, sample a $G = (V, E)$ from the $\mathcal{SW}(n, \alpha)$ distribution and let $G_p$ be the percolation subgraph of $G$. Then, a constant $\underline{p} > 0$ exists such that, if $p < \underline{p}$ then, w.h.p., each connected component of $G_p$ has size at most $\mathcal{O}(\log n)$.*

A key observation to prove Lemma 19 is that, when $\alpha < 1$, the percolation graph of a graph sampled from $\mathcal{SW}(n, \alpha)$ stochastically dominates a cycle with additional Erdős-Rényi random edges. To prove Lemma 19 we thus use a previous result in [3] for this class of random graphs. In particular, we first give an equivalent formulation of Lemma 5.1 in [3], stating that, with constant probability the sequential BFS visit (Algorithm 2) reaches $\Omega(\log n)$ nodes within $\mathcal{O}(\log n)$ rounds. We then consider a *parallel* BFS visit (see Algorithm 4 in Section 6.1) and show an equivalent formulation of Lemma 5.2 in [3], stating that, w.h.p., the parallel BFS visit starting with $\Omega(\log n)$ nodes reaches a constant fraction of nodes within $\mathcal{O}(\log n)$ rounds.

The full proof of Lemma 19 is given in Section 6.1, while the proof of Lemma 20 is omitted, since it proceeds along the very same lines as the proof of Lemma 17, which does not depends on the value of $\alpha$.

### 6.1 Proof of Lemma 19

We first prove the following fact.

▷ Claim 21. Under the hypotheses of Lemma 19, a constant $c \in (0, 1)$ (depending only on $\alpha$) exists such that, for any $u, v \in V$,

$$\mathbf{Pr}\left((u, v) \in E_p\right) \geqslant \frac{pc}{n}.$$

Proof. The normalizing constant $C(\alpha, n)$ (see Definition 1) can be upper bounded as follows

$$C(\alpha, n) = 2\sum_{x=2}^{n/2} \frac{1}{x^\alpha} \leqslant \frac{2}{2^\alpha} + \int_2^{n/2} \frac{2}{x^\alpha} dx \leqslant 2^{1-\alpha} + \frac{2}{1-\alpha}\left(\frac{n}{2}\right)^{1-\alpha}.$$

Hence, a constant $c \in (0,1)$ (depending only on $\alpha$) exists such that

$$\begin{aligned}\mathbf{Pr}\left((u,v) \in E_p\right) &= \frac{p}{C(\alpha,n)d(u,v)^\alpha} \geqslant \frac{p}{C(\alpha,n)(n/2)^\alpha} \\ &\geqslant \frac{p}{2^{1-\alpha}(n/2)^\alpha + n/(1-\alpha)} \geqslant \frac{p \cdot c}{n}.\end{aligned}$$

◁

The above fact proves that, when $\alpha < 1$, the percolation subgraph $G_p$ of a graph sampled from $\mathcal{SW}(n,\alpha)$ stochastically dominates a cycle with additional Erdős-Rényi random edges. To prove Lemma 19 we thus use a previous result in [3] on such class of random graphs. In particular, we first give an equivalent formulation of Lemma 5.1 in [3], that states that, with constant probability the sequential BFS visit (Algorithm 3) reaches $\Omega(\log n)$ nodes within $\mathcal{O}(\log n)$ rounds. We then consider a *parallel*-BFS visit (Algorithm 4) and give an equivalent formulation of Lemma 5.2 in [3], that states that, w.h.p., the parallel-BFS visit starting with $\Omega(\log n)$ nodes reaches a constant fraction of nodes within $\mathcal{O}(\log n)$ rounds.

We also introduce a slightly different version of the BFS visit of Algorithm 4, where we have also a set $R_0$ of removed nodes in input.

**Algorithm 3** BFS visit of $G_p$.

**Input**: the subgraph $G_p = (V, E_p)$, an initiator $s \subseteq V$, a set of removed nodes $R_0 \subseteq V$.
$Q = \{s\}$
$R = R_0$
**while** $Q \neq \emptyset$ **do**
    $w = \texttt{dequeue}(Q)$
    $R = R \cup \{w\}$
    **for** each neighbor $x$ of $w$ in $G_p$ such that $x \notin R$ **do**
        $\texttt{enqueue}(x, Q)$

▶ **Lemma 22.** *Under the hypothesis of Lemma 19, let $v \in V$ be a node and $c \in (0,1)$ be a constant as in Claim 21. For every $\beta > 0$, $\varepsilon > 0$, and percolation probability $p > \frac{\sqrt{c^2+6c+1}-c-1}{2c} + \varepsilon$, there are positive parameters $k$ and $\gamma$ (depending only on $\varepsilon$, $c$ and $p$) such that the following holds: the BFS visit (Algorithm 3) with input $G_p$, $v$, and a set $R_0$ with $|R_0| \leqslant \log^4 n$, with probability $\gamma$, a time $\tau_1 = \mathcal{O}(\log n)$ exists such that*

$$|(R \setminus R_0) \cup Q| \geqslant n/k \qquad OR \qquad |Q| \geqslant \beta \log n\,.$$

**Algorithm 4** Parallel BFS visit of $G_p$.

**Input**: the subgraph $G_p = (V, E_p)$, a set of initiators $I_0 \subseteq V$, a set of removed nodes $R_0 \subseteq V$
$Q = I_0$
$R = R_0$
**while** $Q \neq \emptyset$ **do**
    $A = R \cup Q$
    $X = \text{neighbors}(Q)$
    $Q' = Q$
    $Q = \emptyset$
    **while** $Q' \neq \emptyset$ **do**
        $w = \texttt{dequeue}(Q')$
        $R = R \cup \{w\}$
        **for** each $x \in X$ **do**
            $\texttt{enqueue}(x, Q)$

▶ **Lemma 23.** *Under the hypothesis of Lemma 19, let $c \in (0,1)$ be a constant as in Claim 21. For every $\varepsilon > 0$ and percolation probability $p > \frac{\sqrt{c^2+6c+1}-c-1}{2c} + \varepsilon$, there are positive parameters $k$, $\beta$ (depending only on $c$, $p$ and $\varepsilon$) such that the following holds: For any pair of sets $I_0, R_0 \subseteq V$, with $|I_0| \geqslant \beta \log n$ and $|R_0| \leqslant \log^4 n$, in the parallel BFS-visit (Algorithm 4) with input $G_p$, $I_0$, and $R_0$, with probability at least $1 - 1/n$, a time $\tau_2 = \mathcal{O}(\log n)$ exists such that*

$$|(R \setminus R_0) \cup Q| \geqslant n/k.$$

Now we are ready to prove Lemma 19.

**Proof of Lemma 19.** Let $c$ be as in Claim 21 and $\overline{p} = \frac{\sqrt{c^2+6c+1}-c-1}{2c}$. Let $p > \overline{p} + \varepsilon$, for an arbitrarily-small constant $\varepsilon > 0$, and let $\beta > 0$ be the constant in Lemma 23 and $k$ be the constant in Lemmas 22 and 23. We consider the following process, where we initialize $R_0 = \emptyset$ and $\tau_1 = \mathcal{O}(\log n)$ is as in Lemma 22.

1. Consider a node $v \in V \setminus R_0$.
2. From $v$, perform a sequential-BFS visit (Algorithm 2), with input $G_p$, $v$, and $R_0$, for $\tau_1$ while loops and add to $R_0$ the sets $Q$ and $R$ as they are at the end of the $\tau_1$-th iteration of the while loop ($R_0 = R_0 \cup R \cup Q$).
3. If $|Q| \geqslant \beta \log n$ or $|Q \cup R| \geqslant n/k$, interrupt the process.
4. Restart from 1.

Let $\gamma > 0$ be the constant in Lemma 22. We prove that the process above terminates within $\sigma = \log_{1-\gamma}(n)$ iterations, w.h.p.

First we notice that, at each iteration of the process, the set $R_0$ grows, w.h.p., at most of size $\mathcal{O}(\log^2 n)$, since each node in $G_p$ has degree at most $\mathcal{O}(\log n)$, w.h.p. (Claim 6) and so, in $\tau_1$ iteration of the parallel-BFS, at most $\mathcal{O}(\log^2 n)$ nodes will be reached by $v$. This implies that, at each iteration $i \leqslant \sigma$ of the process $|R_0| = \mathcal{O}(\log^3 n)$, w.h.p.

From Lemma 22 it follows that,a sequential-BFS with in input $G_p$, $v$ and $R_0$ with $|R_0| = \mathcal{O}(\log^3 n)$ is such that, at the end of $\tau_1$-th iteration

$$\mathbf{Pr}\left(|Q| \geqslant \beta \log n \text{ or } |R \cup Q| \geqslant n/k\right) \geqslant \gamma > 0\,.$$

Therefore, the probability that the process exceeds $\sigma$ iterations is at most $(1-\gamma)^\sigma \leqslant 1/n$.

So, w.h.p., a node $v$ exists such that the sequential-BFS starting from $v$, after $\mathcal{O}(\log n)$ steps, satisfies at least one of the two conditions: i) $|Q| \geqslant \beta \log n$ or ii) $|Q \cup R| \geqslant n/k$.

If ii) holds, the lemma is proven. Indeed, w.h.p. we have the existence of a node $v$ such that there is a set of $\Omega(n)$ nodes at distance at most $\mathcal{O}(\log n)$ from $v$.

If i) holds, it suffices to perform a sequential-BFS (Algorithm 2) with in input $G_p$, $I_0 = Q$ and $R_0$ and apply Lemma 23 to claim that such BFS reaches at least $\Omega(n)$ nodes in $\mathcal{O}(\log n)$ steps. ◀

## References


1 Noga Alon, Benjamini Itai, and Stacey Alan. Percolation on finite graphs and isoperimetric inequalities. *Annals of Probability*, 32:1727–1745, 2004.

2 Luca Becchetti, Andrea Clementi, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi. Bond percolation in small-world graphs with power-law distribution. *arXiv preprint*, 2022. arXiv:2205.08774.

**3** Luca Becchetti, Andrea E. F. Clementi, Riccardo Denni, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi. Percolation and epidemic processes in one-dimensional small-world networks - (extended abstract). In *Proceedings of LATIN 2022: the 15th Latin American Symposium*, volume 13568 of *Lecture Notes in Computer Science*, pages 476–492. Springer, 2022. doi:10.1007/978-3-031-20624-5_29.

**4** Itai Benjamini and Noam Berger. The diameter of long-range percolation clusters on finite cycles. *Random Struct. Algorithms*, 19(2):102–111, 2001. doi:10.1002/rsa.1022.

**5** Marek Biskup. On the scaling of the chemical distance in long-range percolation models. *The Annals of Probability*, 32(4):2938–2977, 2004. doi:10.1214/009117904000000577.

**6** Marek Biskup. Graph diameter in long-range percolation. *Random Structures & Algorithms*, 39(2):210–227, 2011. doi:10.1002/rsa.20349.

**7** Duncan S. Callaway, John E. Hopcroft, Jon Kleinberg, Mark E. J. Newman, and Steven H. Strogatz. Are randomly grown graphs really random? *Phys. Rev. E*, 64:041902, September 2001. doi:10.1103/PhysRevE.64.041902.

**8** Wei Chen, Laks Lakshmanan, and Carlos Castillo. Information and influence propagation in social networks. *Synthesis Lectures on Data Management*, 5:1–177, October 2013. doi:10.2200/S00527ED1V01Y201308DTM037.

**9** Hyeongrak Choi, Mihir Pant, Saikat Guha, and Dirk Englund. Percolation-based architecture for cluster state creation using photon-mediated entanglement between atomic memories. *npj Quantum Information*, 5(1):104, 2019.

**10** Easley David and Kleinberg Jon. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, USA, 2010.

**11** Michele Garetto, Weibo Gong, and Donald F. Towsley. Modeling malware spreading dynamics. In *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA, March 30 - April 3, 2003*, pages 1869–1879. IEEE Computer Society, 2003. doi:10.1109/INFCOM.2003.1209209.

**12** Anna R. Karlin, Greg Nelson, and Hisao Tamaki. On the fault tolerance of the butterfly. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of Computing*, pages 125–133, 1994.

**13** David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11(4):105–147, 2015. doi:10.4086/toc.2015.v011a004.

**14** Harry Kesten. The critical probability of bond percolation on the square lattice equals 1/2. *Communications in mathematical physics*, 74(1):41–59, 1980.

**15** Jon Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

**16** Alexander Kott and Igor Linkov. *Cyber resilience of systems and networks*. Springer, 2019.

**17** Rémi Lemonnier, Kevin Seaman, and Nicolas Vayatis. Tight bounds for influence in diffusion networks and application to bond percolation and epidemiology. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 846–854, Cambridge, MA, USA, 2014. MIT Press.

**18** Cristopher Moore and Mark E. J. Newman. Epidemics and percolation in small-world networks. *Physical Review E*, 61(5):5678, 2000.

**19** Cristopher Moore and Mark E. J. Newman. Exact solution of site and bond percolation on small-world networks. *Phys. Rev. E*, 62:7059–7064, November 2000. doi:10.1103/PhysRevE.62.7059.

**20** Charles M. Newman and Lawrence S. Schulman. One dimensional $1/|j-i|^s$ percolation models: The existence of a transition for $s \leqq 2$. *Communications in Mathematical Physics*, 104(4):547–571, 1986.

**21** Mark E. J. Newman and Duncan J. Watts. Scaling and percolation in the small-world network model. *Physical review E*, 60(6):7332, 1999.

**22** Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Rev. Mod. Phys.*, 87:925–979, August 2015. doi:10.1103/RevModPhys.87.925.

23 Vinod K.S. Shante and Scott Kirkpatrick. An introduction to percolation theory. *Advances in Physics*, 20(85):325–357, 1971. doi:10.1080/00018737100101261.

24 Wei Wang, Ming Tang, H Eugene Stanley, and Lidia A Braunstein. Unification of theoretical approaches for epidemic spreading on complex networks. *Reports on Progress in Physics*, 80(3):036603, February 2017. doi:10.1088/1361-6633/aa5398.

25 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, 1998.

26 Tongfeng Weng, Michael Small, Jie Zhang, and Pan Hui. Lévy walk navigation in complex networks: A distinct relation between optimal transport exponent and network dimension. *Scientific Reports*, 5(1):17309, 2015.

# Computing Temporal Reachability Under Waiting-Time Constraints in Linear Time

**Filippo Brunelli**
Université Paris Cité, Inria, CNRS, IRIF, Paris, France

**Laurent Viennot**
Université Paris Cité, Inria, CNRS, IRIF, Paris, France

## Abstract

This paper proposes a simple algorithm for computing single-source reachability in a temporal graph under waiting-time constraints, that is when waiting at each node is bounded by some time constraints. Given a space-time representation of a temporal graph, and a source node, the algorithm computes in linear-time which nodes and temporal edges are reachable through a constrained temporal walk from the source.

**2012 ACM Subject Classification** Theory of computation → Dynamic graph algorithms; Mathematics of computing → Graph algorithms

**Keywords and phrases** temporal reachability, temporal graph, temporal path, temporal walk, waiting-time constraints, restless temporal walk, linear time

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.4

**Related Version** *Extended Version*: https://hal.inria.fr/hal-03864725v2

**Funding** This work was supported by the French National Research Agency (ANR) through project Tempogral with reference number ANR-22-CE48-0001.

## 1 Introduction

Reachability, that is connectivity through a path, is a fundamental notion in graphs. There exist simple and elegant algorithms that determine efficiently which nodes or edges can be reached from a given source node, and finding optimal paths realizing such reachability. On the other hand, temporal graphs, where edges evolve over time, offer a richer variety of temporal connectivity, especially when considering waiting constraints as we discuss below. By focusing on reachability without concern for any optimization criterion, we aim at designing a simple algorithm under waiting constraints.

### Temporal graphs

Temporal graphs arose with the need to better model contexts where the appearance of interactions or connections depends on time, such as epidemic propagation or transport networks. Starting with the work on time-dependent networks [8] and the telephone problem [5], the discrete time version of temporal graphs we consider here was already investigated in [2, 16, 18] and introduced later in various contexts ranging from social interactions to mobile networks and distributed computing (see e.g. [6, 14, 13]). This classical point-availability model of temporal graph is the following. The availability of an edge $(u, v)$ at time $\tau$ is modeled by a temporal edge $e = (u, v, \tau, \lambda)$. It represents the possibility to traverse the edge from $u$ at time exactly $\tau$ with arrival in $v$ at time $\tau + \lambda$. We refer to $\tau$ and $\tau + \lambda$ as the departure time and arrival time of $e$ respectively, while $\lambda > 0$ is called the travel time of $e$. Notice that we consider travel time to be strictly positive, which is a natural assumption when it comes down to application like, for example, transport networks. A temporal walk can


© Filippo Brunelli and Laurent Viennot;
licensed under Creative Commons License CC-BY 4.0

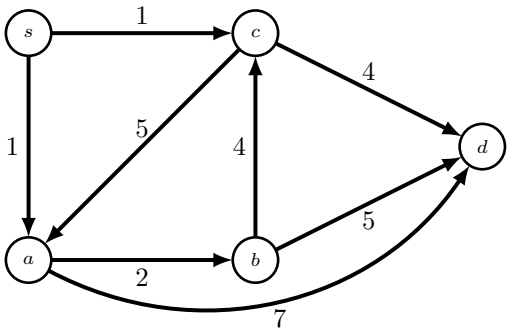



**Figure 1** A temporal graph with waiting constraints. Each temporal edge in the picture is labeled with its departure time and has travel time one, each node has minimum waiting-time $\alpha = 0$ and maximum waiting-time $\beta = 1$. The only temporal edge entering $d$ and reachable from $s$ is $(a, d, 7, 1)$ through the temporal walk $(s, a, 1, 1), (a, b, 2, 1), (b, c, 4, 1), (c, a, 5, 1), (a, d, 7, 1)$. Indeed, following any of the two other edges $(b, d, 5, 1)$ and $(c, d, 4, 1)$ would require to wait $2 > \beta$ units of time either at $b$ (after edge $(a, b, 2, 1)$) or $c$ (after edge $(s, c, 1, 1)$).

then be defined as a sequence of temporal edges such that each temporal edge arrives at the departing node of the next one, and the arrival time of each temporal edge is less or equal to the departure time of the next one. The inequality means that it is possible to wait at the node in-between two consecutive temporal edges. In particular, the time elapsed between the arrival time of an edge and the departure of the next one represents the amount of time spent waiting at the node. We distinguish such a walk from a temporal path, which is a temporal walk visiting at most once any node.

Without waiting constraints, that is when waiting at a node is unrestricted, numerous works have investigated single-source temporal path computation with a primary focus on earliest arrival. After several works inspired by Dijkstra algorithm (see e.g. [2, 3, 16, 18]), a simple and elegant linear-time algorithm for earliest arrival time was first claimed in [20] with a similar algorithm as [10] through a single scan of temporal edges ordered by non-decreasing departure time. Assuming strictly positive travel times is important here as it ensures that the temporal edges of any temporal path appear in order during this scan. These algorithms indeed focus their target on temporal paths rather than walks, since unrestricted waiting allows to transform any walk into a path by waiting at nodes instead of performing any loops. In this setting they allow to determine which nodes or edges are reachable. However, we consider the following more general model.

### Waiting constraints

We consider temporal graphs subject to waiting constraints. In such graphs, during a temporal walk, it is not possible to wait at a node less than $\alpha$ time or more than $\beta$ time, before moving to another node. Such constraints can be used to model, for example, preferences of a user in a public transport network, or to take into account incubation time and recovery time of a disease in a temporal network of contacts.

We focus on the following rechability problem. We say that a temporal edge $e = (u, v, \tau, \lambda)$ is reachable from a node $s$ if there exists a temporal walk from $s$ ending with $e$ and respecting waiting constraints. The *reachability problem* consists in identifying all the temporal edges that are reachable from a given source node $s$ (see Figure 1 for an example). Note that we can easily compute which nodes are reachable from this. Moreover, this problem generalizes the

single-source earliest arrival time problem: indeed, given the set of the $s$-reachable edges, a linear scan allows to identify for each node $v$ the $s$-reachable edge with head $v$ that has lowest arrival time, and which corresponds to the earliest arrival time at $v$. Computing the number of nodes reachable from a given source can be interesting when measuring connectivity properties of a temporal network [9]. Reachability of edges additionally provides information about all times at which it is possible to reach such nodes. This is related to the profile problem [11], which consists in computing a function that for each possible departure time from the source return the earliest arrival time towards the destinations.

Waiting constraints significantly modify temporal connectivity. Most strikingly, it has been proved that the computation of temporal paths in this setting becomes NP-hard [7]. While unrestricted waiting makes reachability through temporal paths or walks equivalent, this result moves the interest to the sole case of temporal walks when dealing with bounded waiting. It is thus necessary to design algorithms following a different approach.

In a recent break-through, [1] proposes an algorithm computing single-source optimal temporal walks under waiting-time constraints in $O(M \log M)$ time, where $M$ is the total number of temporal edges in the graph. It computes walks that optimize a linear combination of the most classical criteria, and can thus solve the reachability problem as well. The algorithm is quite involved. It relies on first transforming the temporal graph so as to zero all travel times and then performing a Dijkstra computation for each time instant when a temporal edge departs or arrives. More precisely, it first builds an equivalent temporal graph where all edges have zero travel time. This is done by adding a dummy node with appropriate waiting restrictions for each temporal edge. Note that this can considerably increase the number of nodes. Then, it scans time instants when temporal edges occur in increasing order. For each time instant $t$, a static directed graph is constructed and a Dijkstra computation allows to update the reachability of nodes with temporal edges up to time $t$. It feels natural to investigate whether in the context of bounded waiting it is possible to develop an easier and more efficient method to solve the simpler reachability problem.

## Contribution

We develop an algorithm to solve the reachability problem in temporal graphs subject to waiting constraints. The main strength of the algorithm is its simplicity, which comes with no downplay in efficiency, since it runs in linear time. It thus improves by a factor $\log M$ the state of the art in the setting of positive travel times. The algorithm performs a linear scan of the list of temporal edges, in a spirit similar to [10]. In this case, however, the algorithm requires in input not one, but two, sorted lists, one containing the temporal edges sorted by departure time and the other by arrival time. Exploiting these two lists is a new technique for handling waiting constraints that could be useful for computing optimum temporal walks or other temporal connectivity problems. Interestingly, this representation of the temporal graph through two lists is closely related to the more classical "space-time" (or "time-expanded") graph [17, 18, 19, 15, 14] where each node is split into node events, one for each time where a temporal edge arrives to it or departs from it, and each temporal edge is turned into an arc between two node events. If the input is given as a list of temporal edges, the two appropriate lists can be obtained in $O(M \log M)$ time. However, our algorithm runs in linear time when given a space-time representation as input since two appropriate lists can easily be obtained from a topological ordering of the corresponding static directed graph. The positive travel time assumption can be loosened to a more general acyclic setting as described in a follow-up paper [4].

The paper is organized as follows. In Section 2 we define the main notions. In Section 3 we present our main algorithmic result. Finally, in Section 4 we show how to adapt our algorithm in order to take as input a space-time representation of a temporal graph.

## 2 Preliminary definitions

A *temporal graph* is a tuple $G = (V, E, \alpha, \beta)$, where $V$ is the set of *nodes*, $E$ is the set of temporal edges and $\alpha, \beta \in [0, +\infty]^V$ are minimum and maximum waiting-times at each node. A *temporal edge* $e$ is a quadruple $(u, v, \tau, \lambda)$, where $u \in V$ is the *tail* of $e$, $v \in V$ is the *head* of $e$, $\tau \in \mathbb{R}$ is the *departure time* of $e$, and $\lambda \in \mathbb{R}_{>0}$ is the *travel time* of $e$. We also define the *arrival time* of $e$ as $\tau + \lambda$, and we let $dep(e) = \tau$ and $arr(e) = \tau + \lambda$ denote the departure time and arrival time of $e$ respectively. For the sake of brevity, we often say edge instead of temporal edge. We let $n = |V|$ and $M = |E|$ denote the number of nodes and edges respectively.

Given a temporal graph $G = (V, E, \alpha, \beta)$ a *walk* $Q$ from $u$ to $v$, or a *uv-walk* for short, is a sequence of temporal edges $\langle e_1 = (u_1, v_1, \tau_1, \lambda_1), \ldots, e_k = (u_k, v_k, \tau_k, \lambda_k)\rangle \subseteq E^k$ such that $u = u_1$, $v_k = v$, and, for each $i$ with $1 < i \leq k$, $u_i = v_{i-1}$ and $a_{i-1} + \alpha_{u_i} \leq \tau_i \leq a_{i-1} + \beta_{u_i}$ where $a_{i-1} = \tau_{i-1} + \lambda_{i-1}$ is the arrival time of $e_{i-1}$. Note that the waiting time $\tau_i - a_{i-1}$ at node $u_i$ is constrained to be in the interval $[\alpha_{u_i}, \beta_{u_i}]$. Note that since travel times are positive, such a walk is *strict* in the sense that $\tau_{i-1} < \tau_i$ for $1 < i \leq k$ as the constraint $a_{i-1} + \alpha_{u_i} \leq \tau_i$ implies $\tau_i \geq a_{i-1} = \tau_{i-1} + \lambda_{i-1} > \tau_{i-1}$ for $\lambda_{i-1} > 0$. The *departing time* $dep(Q)$ of $Q$ is defined as $\tau_1$, while the *arrival time* $arr(Q)$ of $Q$ is defined as $\tau_k + \lambda_k$. We say that a temporal edge $e = (x, y, \tau, \lambda)$ *extends* $Q$ when $x = v_k$ and $arr(Q) + \alpha_x \leq \tau \leq arr(Q) + \beta_x$. When $e$ extends $Q$, we can indeed define the walk $Q.e = \langle e_1, \ldots, e_k, e\rangle$ from $u$ to $y$. Moreover, we also say that $e$ extends $e_k$ as it indeed extends any walk $Q$ having $e_k$ as last edge. We also say that an edge $e$ is an *s-reachable edge* whenever there exists an *sv*-walk ending with edge $e$. Let us now introduce some orderings of temporal edges with respect to certain temporal criteria. Given an ordering of the temporal edges $E^{ord}$ we use $e <_{E^{ord}} f$ to denote that $e$ appears before $f$ in $E^{ord}$. We say that an ordering $E^{ord}$ of all the temporal edges is *departure sorted* if the edges are ordered by non-decreasing departure time in $E^{ord}$, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ satisfy $dep(e) < dep(f)$. Similarly, we say that an ordering $E^{ord}$ of all the temporal edges is *arrival sorted* if the edges are ordered by non-decreasing arrival time in $E^{ord}$, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ satisfy $arr(e) < arr(f)$.

Finally, we define the *doubly-sorted representation* of a temporal graph $(V, E, \alpha, \beta)$ as a data-structure with two lists $(E^{dep}, E^{arr})$, containing $|E|$ quadruples each, representing all temporal edges in $E$, where $E^{dep}$ is a list sorted by non-decreasing departure time and $E^{arr}$ is a list sorted by non-decreasing arrival time. Moreover, we assume that we have implicit pointers between the two lists, that link each quadruple of one list to the quadruple representing the same temporal edge in the other list.

Without loss of generality, we can restrict our attention to nodes appearing as head or tail of at least one temporal edge and we thus assume $|V| = O(|E|)$. An algorithm is said to be linear in time and space when it runs in $O(|E|)$ time and uses $O(|E|)$ space. Given a doubly sorted representation $(E^{dep}, E^{arr})$, we also assume that we are given for each node $v$ the list $E^{dep}_v$ of pointers to temporal edges with tail $v$ ordered by non-decreasing departure time, as it can be computed in linear time and space from $E^{dep}$ through bucket sorting. We assume that each list $E^{arr}$, $E^{dep}$, or $E^{dep}_v$ is stored in an array $T$ such that each element $T[i]$ can be accessed directly through its index $i \in [1, |T|]$ in constant time. Given two indexes $i \leq j$, we also let $T[i : j]$ denote the sub-array of elements of $T$ with index in $[i, j]$.

## 3 A linear-time Algorithm to compute reachability

In this section we will provide our main result: an algorithm that solves in linear time and space the reachability problem, which is defined as follows.

> Singles-Source Reachability Problem. Given a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node $s$, compute the set of all temporal edges that are $s$-reachable.

In the following we will assume to be given a doubly-sorted representation $(E^{dep}, E^{arr})$ of the temporal graph. We design an algorithm which mainly consists in scanning linearly edges in $E^{arr}$ while updating the set $A_v$ of $s$-reachable edges terminating $sv$-walks in the temporal graph resulting from the edges read so far. To help identifying edges that will appear in such walks in next iterations, we also mark edges that extend these walks.

We now describe more precisely how edges are scanned and marked as formalized in Algorithm 1. We first build the lists $E_v^{dep}$ of temporal edges with tail $v$ by bucket sorting $E^{dep}$ at Line 1. We then identify the $s$-reachable edges as follows. We linearly scan $E^{arr}$. In the temporal graph resulting from the temporal edges read up to edge $e = (u, v, \tau, \lambda) \in E^{arr}$, the only walks from $s$ that have not been considered yet must contain $e$, and must have it as last edge as $E^{arr}$ is sorted by non-decreasing arrival time. If its tail $u$ is $s$, or if $e$ is marked, then we know that there exists a walk from $s$ to its head $v$. In that case, we add edge $e$ to $A_v$ at Line 9, and we then mark edges that extend $e$, that is edges in $E_v^{dep}$ with departure time in $[a + \alpha_v, a + \beta_v]$, since the arrival time of $e$ is $a = \tau + \lambda$. These edges appear consecutively in $E_v^{dep}$ which is processed linearly as walks from $s$ to $v$ are identified. This process is done in Lines 10-14 in Algorithm 1, starting from the index $p_v$ of the last processed edge in $E_v^{dep}$, and such edges $f$ that extend $e$ are marked at Line 13 before updating $p_v$. Moreover, we use classical parent pointers to be able to compute an $sv$-walk for each $s$-reachable edge with head $v$. Each parent pointer $P[f]$ of an edge $f$ is initially set to a null value $\perp$ at Line 6. Whenever we mark edge $f$, that extend the currently scanned edge $e$, we set the parent pointer of $f$ to $e$. If $f$ is an $s$-reachable edge at $v$, we can then get an $sv$-walk by following the parent pointer $P[f], P[P[f]], \ldots$.

▶ **Theorem 1.** *Given a doubly-sorted representation of a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node $s \in V$, Algorithm 1 computes all $s$-reachable temporal edges in linear time and space.*

**Proof.**

**Correctness.** Let us denote by $G_k = (V, E^{arr}[1 : k], \alpha, \beta)$ the temporal graph induced by the first $k$ temporal edges in $E^{arr}$. We will prove, by induction on $k$, the following two invariants:

**$(I_k^1)$** For every node $v$, $A_v$ contains all $s$-reachable edges with head $v$ in $G_k$.
**$(I_k^2)$** The marked edges are all the edges in $E$ that extend a walk from $s$ in $G_k$.

The correctness of the algorithm will follow from invariant $(I_k^1)$ for $k = |E|$. The invariants are satisfied for $k = 0$ since there are no edges in $G_0$ while the sets $(A_v)_{v \in V}$ of $s$-reachable edges are initially empty and no edge is initially marked.

Now suppose that the two invariants hold for $k - 1$, with $k \geq 1$, and let us prove that they still hold for $k$ after scanning the $k$th edge $e_k = (u, v, \tau, \lambda)$ in $E^{arr}$. To prove $(I_k^1)$ and $(I_k^2)$, we first show that the condition of the if statement at Line 8 is met when $e_k$ is an $s$-reachable edge in $G_k$. It is obviously the case when $u = s$ as $\langle e_k \rangle$ is in $G_k$, or when $e_k$ was

**Algorithm 1** Computing, for each node $v$, the set $A_v$ of all $s$-reachable edges with head $v$.

**Input:** A doubly-sorted representation $(E^{arr}, E^{dep})$ of a temporal graph $G$ with waiting constraints $(\alpha, \beta)$, and a source node $s \in V$.
**Output:** The sets $(A_v)_{v \in V}$ of $s$-reachable edges at each node $v$ sorted by non-decreasing arrival time.

1 For each node $v$, generate the list $E_v^{dep}$ by bucket sorting $E^{dep}$.
2 **For** *each node* $v$ **do**
3  Set $A_v := \emptyset$. `/* Set of s-reachable edges (as a sorted list). */`
4  Set $p_v := 0$. `/* Index of the last processed edge in` $E_v^{dep}$`. */`
5 Set all the edges in $E^{arr}$ as unmarked.
6 Set $P[e] := \perp$ for each edge $e \in E^{arr}$.`/* Parent of` $e$`, initially null. */`
7 **For** *each edge* $e = (u, v, \tau, \lambda)$ *in* $E^{arr}$ **do**
8  **If** $u = s$ *or* $e$ *is marked* **then**
  `/*` $e$ `is s-reachable. */`
9  $A_v := A_v \cup \{e\}$
10  Let $a = \tau + \lambda$ be the arrival time of $e$.
  `/* Process further edges from` $v$ `with dep. time` $\leq a + \beta_v$`: */`
11  Let $l > p_v$ be the first index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \geq a + \alpha_v$ (set $l := |E_v^{dep}| + 1$ if no such index exists).
12  Let $r \geq l$ be the last index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \leq a + \beta_v$ (set $r := l - 1$ if no such index exists).
  `/* Mark unmarked edges with dep. time in` $[a + \alpha_v, a + \beta_v]$`: */`
13  **If** $l \leq r$ **then** mark each edge $f \in E_v^{dep}[l : r]$ and set $P[f] := e$.
14  Set $p_v := r$.
15 **Return** *the sets* $(A_v)_{v \in V}$.

previously marked, as Invariant $(I_{k-1}^2)$ then implies that it extends a walk $Q$ from $s$ in $G_{k-1}$ and that $Q.e_k$ is a walk in $G_k$. The converse also holds: if $e_k$ is an edge of a walk $Q$ from $s$ in $G_k$, then either it is the first edge and we have $u = s$ or the sequence $Q'$ of edges before $e_k$ in $Q$ is a walk in $G_{k-1}$ and $(I_{k-1}^2)$ implies that it is marked.

Note that when $e_k$ appears in a walk $Q$ of $G_k$, it must be the last edge of $Q$ as $E^{arr}$ is sorted by non-decreasing arrival time and edges have positive travel time. This allows to prove $(I_k^1)$: as we assume $(I_{k-1}^1)$, we just have to consider walks from $s$ that are in $G_k$ but not in $G_{k-1}$, that is those containing $e_k$. Since all these walks have $e_k$ as last edge, and $e_k$ is the only edge added to $A_v$ when such walks exist, we can conclude that $(I_k^1)$ holds.

Similarly, to prove $(I_k^2)$ when $(I_{k-1}^2)$ holds, we just have to consider the edges extending a walk $Q$ from $s$ which is in $G_k$ but not in $G_{k-1}$. As discussed above, when such a walk $Q$ exists, $e_k$ is its last edge and the condition of the if statement Line 8 holds. Edges extending such a walk $Q$ are thus those extending $e_k$, that is all edges $f \in E_v^{dep}$ such that $a + \alpha_v \leq dep(f) \leq a + \beta_v$. Note that the ordering of $E_v^{dep}$ implies that these edges are consecutive in $E_v^{dep}$. If no such edges exist, let $l'$ and $r'$ designate the first and last indexes respectively where they are placed in $E_v^{dep}$. To prove $(I_k^2)$, it thus suffices to prove that all edges in $E_v^{dep}[l' : r']$ are marked after scanning $e_k$ and that only edges in $E_v^{dep}[l' : r']$ are marked during the iteration for $e_k$ (if no such edges exist we prove that we mark no edges). Consider the values $l$ and $r$ computed at Lines 11 and 12 respectively. If no edge $f$ extends $e_k$, then we get $r = l - 1$ and no edge is marked. Now, we assume that such edges exist and

that $l'$ and $r'$ are well defined. First assume $l \leq r$ and thus that $l$ was not set to $|E_v^{dep}| + 1$. The choice of $l, r$ then imply $a + \alpha_v \leq dep(E_v^{dep}[l])$ and $dep(E_v^{dep}[r]) \leq a + \beta_v$. We thus have $l' \leq l \leq r \leq r'$ and all marked edges at Line 13 are in $E_v^{dep}[l' : r']$. Moreover, the choice of $r$ indeed then implies $r = r'$. We still need to prove that edges in $E_v^{dep}[l' : l - 1]$ have already been marked. Otherwise, when $r = l - 1$, no edge is marked. This occurs when $p_v \geq r'$ and we then have $l = p_v + 1$. In both cases, it remains to prove that all edges $f \in E_v^{dep}[l' : \min\{l - 1, r'\}]$ have already been marked. This interval is non empty when $l' \leq l - 1$ and thus $p_v = l - 1$ by the choice of $l$. We thus have $p_v \geq \min\{l - 1, r'\}$. Let $i$ be the index of $f$ in $E_v^{dep}$ and consider the iteration $j < k$ when $p_v$ was updated from a value smaller than $i$ to a value $r'' \geq i$ where $l''$ and $r''$ denote the indexes computed for variables $l$ and $r$ respectively during the $j$-th iteration for edge $e_j \in E^{arr}$.

Since $E^{arr}$ is sorted by non-decreasing arrival time, the arrival time $a'$ of $e_j$ satisfies $a' \leq a$ and we thus have $dep(f) \geq a + \alpha_v \geq a' + \alpha_v$. The choice of index $l$ at Line 11 in that iteration thus guarantees that the index $l''$ must satisfy $l'' \leq i$. We thus have $l'' \leq i \leq r''$ and $f$ was marked at Line 13 during the $j$th iteration. This completes the proof of $(I_k^2)$.

We finally prove that the parent pointers allow us to compute for each $s$-reachable edge $f = (u, v, \tau, \lambda)$ with head $v$ an $sv$-walk ending with $f$. If $f \in A_v$ and it is not marked, then $P[f] = \perp$, and we must have $u = s$ as $f$ was added to $A_v$. In this case, $\langle f \rangle$ is an $sv$-walk itself. Now consider the case $f \in A_v$ and $f$ is marked. Consider the iteration $k$ where $f$ was marked. By $(I_{k-1}^2)$ and $(I_k^2)$, $f$ extends a walk from $s$ ending with $e_k$, where $e_k$ is the edge scanned at iteration $k$, and $P[f]$ was then set to $e_k$. This guarantees by a simple induction that, if $P[f] \neq \perp$, by following the parent pointers in classical manner, namely $P[f], P[P[f]], \ldots,$ until $\perp$ is found, it is possible to obtain a walk terminating with edge $f$.

**Complexity analysis.** The preprocessing of $E^{dep}$ and the initialization from Line 1 to Line 6 clearly take linear time. The main for loop scans each temporal edge $e = (u, v, \tau, \lambda)$ in $E^{arr}$ exactly once. For each iteration there are three operations that may require non-constant time: the computation of $l$ and $r$ at Lines 11 and 12, and marking edges in $E_v^{dep}[l, r]$ at Line 13. They all take $O(r - p_v)$ time as $l$ and $r$ can be found by scanning edges in $E_v^{dep}$ from $p_v + 1$. Thanks to the update of the index $p_v$ to $r$, each edge in $E_v^{dep}$ is processed at most once for a total amortized cost of $O(|E_v^{dep}|)$. Overall, this leads to a time complexity of $O(|E| + \sum_{v \in V} |E_v^{dep}|) = O(|E|)$. Algorithm 1 thus runs in linear time. Finally, let us notice that for all nodes $v$, the set $A_v$ has size bounded by the number of temporal edges with head $v$. We thus have $\sum_{v \in V} |A_v| \leq |E|$, and the space complexity of Algorithm 1 is linear. ◀

## 4 Taking a space-time representation as input

Let us first recall the definition of the "space-time" representation [17]. It consists in a transformation of a temporal graph into a static graph by introducing a copy of each node for each possible time instant. Each temporal edge is then turned into a static edge from the two corresponding copies of its tail and head. We consider here a variant where we introduce copies of a node only for time instants corresponding to a departure time of an edge from that node, or an arrival time of an edge to that node, following the approach of [19].

Formally, given a temporal graph $G = (V, E)$, its *space-time representation* is a directed graph $D = (W, F^c \cup F^w)$, where:

- The nodes in $W$ are labeled nodes $v_\tau$, where $v \in V$ refers to a node of $G$ and $\tau$ is a time label. More precisely, $v_\tau \in W$ if and only if there exists a temporal edge in $E$ with tail $v$ and departure time $\tau$ or a temporal edge with head $v$ and arrival time $\tau$. We will also refer to such nodes as *copies of* $v$. Let us denote with $Pred^w(v_\tau)$ the copy of $v$ in $W$ with maximum time label less than $\tau$, if it exists.

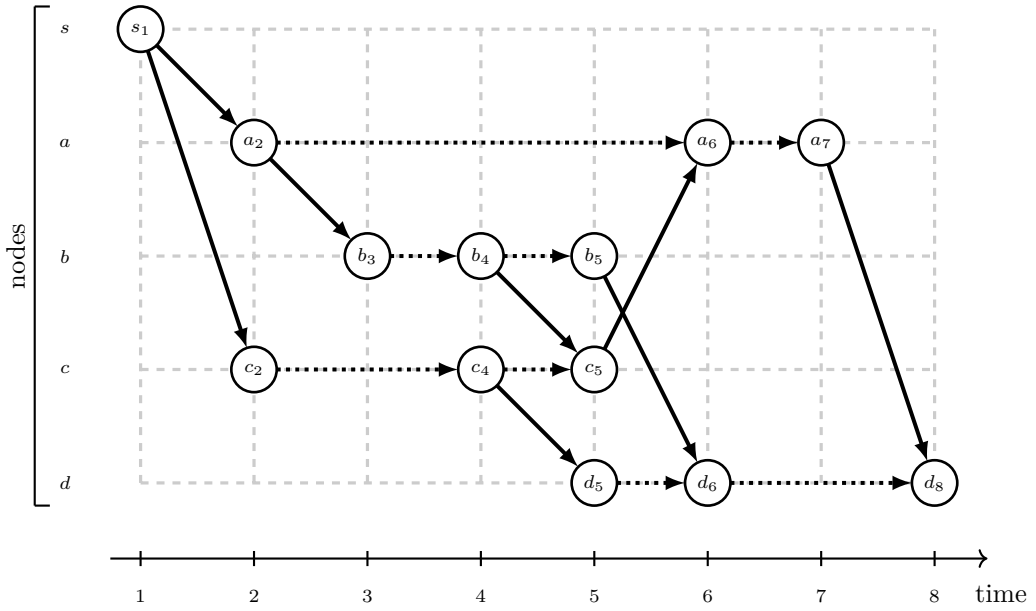



**Figure 2** Space-time representation of the temporal graph of Figure 1. Plain arcs correspond to temporal edges while dotted arcs correspond to waiting at a node. The temporal walk $(s, a, 1, 1), (a, b, 2, 1), (b, c, 4, 1), (c, a, 5, 1), (a, d, 7, 1)$ corresponds to the directed path $s_1, a_2, b_3, b_4, c_5, a_6, a_7, d_8$. Note that the directed path $s_1, a_2, b_3, b_4, b_5, d_6$ does not corresponds to a valid temporal walk, since waiting at node $b$ from time 3 to time 5 violates the constraint $\beta = 1$.

- We distinguish two types of arcs $F^c$ and $F^w$ called connection arcs and waiting arcs respectively. The set $F^c$ contains an arc $(u_\tau, v_{\tau+\lambda})$ for each temporal edge $e = (u, v, \tau, \lambda) \in E$. These arcs represent a temporal connection between nodes in $V$ and are called *connection arcs*. Note that each arc $(v_\tau, w_\nu)$ in $F^c$ satisfies $\tau < \nu$, since travel times are positive. The set $F^w$ is defined to contain an arc $(Pred^w(v_\tau), v_\tau)$ for each $v \in V$ and for each copy $v_\tau$ of $v$ such that $Pred^w(v_\tau)$ is defined. These arcs represent the possibility to wait at a node in $v \in V$ during a walk in $G$ and are called *waiting arcs*. Note that each arc $(v_\tau, v_\nu)$ in $F^w$ satisfies $\tau < \nu$.

The main property of this representation is that any temporal walk $Q$ corresponds to a directed path in the representation using arcs in $F^c$ corresponding to temporal edges of $Q$ plus waiting arcs in $F^w$ each time the walk waits at a node (see Figure 2 for an example). Note that the converse is also true in the unrestricted waiting setting but not with waiting-time constraints.

We will now show that Algorithm 1 runs correctly when $E^{arr}$ and $E^{dep}$ satisfy weaker requirements and how to compute such lists from a space-time representation.

Let us introduce some orderings of temporal edges with respect to certain temporal criteria. We say that an ordering $E^{ord}$ of the edges of a temporal graph $G$ is *walk-respecting* when the edges of any walk $Q$ in $G$ appear in order in $E^{ord}$. Equivalently, $E^{ord}$ is walk-respecting when for any pair $e, f \in E$ of edges such that $f$ extends $e$, then $e <_{E^{ord}} f$, where $e <_{E^{ord}} f$ means that $e$ appears before $f$ in $E^{ord}$. Moreover, we say that an ordering $E^{ord}$ of all the temporal edges is node-departure sorted if all edges departing from the same node are ordered by non-decreasing departure time in $E^{ord}$, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ have same tail and satisfy $dep(e) < dep(f)$. Similarly, we say that an ordering $E^{ord}$

of all the temporal edges is node-arrival sorted if all edges arriving to the same node are ordered by non-decreasing arrival time in $E^{ord}$, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ have same head and satisfy $arr(e) < arr(f)$.

Let us consider a temporal graph $G$ and its temporal edges $E$. Let $(E^{dep}, E^{arr})$ be two lists representing all temporal edges in $E$, where $E^{dep}$ is node-departure sorted, and $E^{arr}$ is walk-respecting and node-arrival sorted. Notice that these hypothesis are weaker than the doubly-sorted representation we defined and used earlier. Indeed, if $E^{arr}$ is sorted by non-decreasing arrival time, then it is trivially node-arrival sorted. It is also walk-respecting since we are assuming positive travel time of the temporal edges, thus the edges of a temporal walk $Q$ have strictly increasing arrival times. On the other side, a simple example can prove that the opposite does not hold. Let $e = (s, u, \tau_1, \lambda_1)$ and $f = (s, v, \tau_2, \lambda_2)$ be two temporal edges such that $arr(e) < arr(f)$. Then $\{f, e\}$ is a node-arrival and walk-respecting sorted list, but it is not sorted by non-decreasing arrival time. We can now state the following.

▷ Claim 2. Given two lists $(E^{dep}, E^{arr})$, where $E^{dep}$ is node-departure sorted, and $E^{arr}$ is walk-respecting and node-arrival sorted, that represent a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$, and a source node $s \in V$, Algorithm 1 computes all $s$-reachable temporal edges in linear time and space.

We sketch a proof of Claim 2 by going through the key points that exploited the order of the lists of the proof of correctness in Theorem 1:

- In the preprocessing, it is still possible to compute in linear time, for each node $v$, the list $E^{dep}_v$ by bucket sorting $E^{dep}$.
- Let $e_k$ be the edge scanned during the $k$-th iteration. Then, if $e_k$ appears in a walk $Q$ in $G_k$ it is still its last edge. The reason is that if it appears in a previous position in $Q$, then $Q$ would contradict the walk-respecting hypothesis of $E^{arr}$.
- When proving that the algorithm correctly marks edges in $E^{dep}_v[l' : r']$, we used the non-decreasing arrival time property of $E^{arr}$ to leverage that the edges entering $v$ are scanned in $E^{arr}$ by non-decreasing arrival time. This property actually coincides with the node-arrival definition.

We now show that an appropriate pair of lists $(E^{dep}, E^{arr})$ can easily be computed from a space-time representation.

▷ Claim 3. Given the space-time representation $D = (W, F^c \cup F^w)$ of a temporal graph $G = (V, E)$, it is possible to compute in linear time ans space two lists $(E^{dep}, E^{arr})$, where $E^{dep}$ is node-departure sorted, and $E^{arr}$ is walk-respecting and node-arrival sorted.

In order to prove Claim 3 we provide a simple algorithm based on Kahn's algorithm [12]. Indeed, because of the positive travel assumption, $D$ is a directed acyclic graph. It is then possible to use Kahn's algorithm to compute a topological ordering of $D$ in linear time that is an ordering $W^{ord}$ of $W$ such that $u_\tau <_{W^{ord}} v_\nu$ for all arcs $(u_\tau, v_\nu) \in F^c \cup F^w$.

Let us see how to compute a list of temporal edges $E^{arr}$ that is node-arrival and walk-respecting sorted from such a topological ordering $W^{ord}$ of $D$ (see Figure 3 for an example). We start with an empty list $E^{arr}$. Then, for each node $v_\nu \in W^{ord}$, from the first one to the last one, we consider its incoming arcs $(u_\tau, v_\nu)$ in $F^c$. For each of such arc $(u_\tau, v_\nu)$, we append to $E^{arr}$ the temporal edge $(u, v, \tau, \nu - \tau)$. The list we obtain this way is:

- Walk-respecting sorted: A temporal walk $Q$ in $G$ corresponds to a path $P$ in $D$. Moreover, the time labels of the nodes in $P$ are strictly increasing. Thus the topological order guarantees that we consider the arcs in $P \cap F^c$ in the same order as the corresponding temporal edges appear in $Q$.

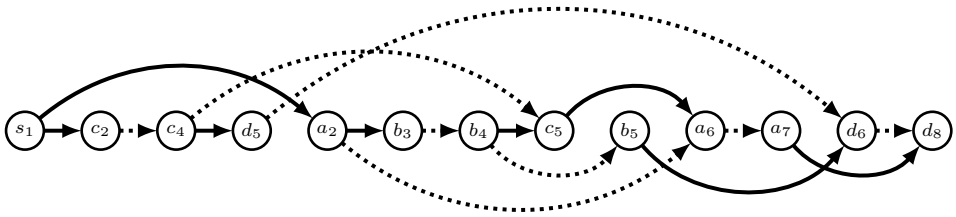



**Figure 3** The topological order $s_1, c_2, c_4, d_5, a_2, b_3, b_4, c_5, b_5, a_6, a_7, d_6, d_8$ of the space-time representation of Figure 2 leads to the node-arrival sorted and walk-respecting ordering $E^{arr} = (s, c, 1, 1)$, $(c, d, 4, 1)$, $(s, a, 1, 1)$, $(a, b, 2, 1)$, $(b, c, 4, 1)$, $(c, a, 5, 1)$, $(b, d, 5, 1)$, $(a, d, 7, 1)$. It also results in the node-departure sorted and walk-respecting ordering $E^{dep} = (s, a, 1, 1)$, $(s, c, 1, 1)$, $(c, d, 4, 1)$, $(a, b, 2, 1)$, $(b, c, 4, 1)$, $(c, a, 5, 1)$, $(b, d, 5, 1)$, $(a, d, 7, 1)$.

- Node-arrival sorted: Since there is a path connecting the copies of each node $v \in V$ through increasing time labels, we are guaranteed to extract each copy by increasing time label. Thus the edges entering $v$ will be considered and appended to $E^{arr}$ by non-decreasing arrival time.

A list $E^{dep}$ of temporal edges which is node-departure sorted (and walk respecting) can be similarly obtained in linear time by scanning out-arcs of each node in the topological ordering instead of in-arcs. As a consequence of Theorem 1, Claim 2 and Claim 3, we thus obtain:

▶ **Theorem 4.** *Given a doubly-sorted representation, or a space-time representation, of a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node $s \in V$, it is possible to compute all $s$-reachable temporal edges in linear time and space.*

## 5 Conclusion

We provided an algorithm that, given in input a space-time representation of a temporal graph with waiting constraints, computes in linear time and space all the reachable edges from a given source. In particular, this also solves the single-source earliest arrival time problem. We are working on extending this technique for computing single-source optimal temporal walks optimizing classical criteria such as shortest duration or number of edges [4].

A future line of work consists in considering a more flexible model in the context of applications to public transport networks, for example by also allowing *footpaths* arcs that are available at any point in time.

## References

1 Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Appl. Netw. Sci.*, 5(1):73, 2020.

2 Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996.

3 Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electron. Notes Theor. Comput. Sci.*, 92:3–15, 2004.

4 Filippo Brunelli and Laurent Viennot. Minimum-cost temporal walks under waiting-time constraints in linear time. *CoRR*, abs/2211.12136, 2022. doi:10.48550/arXiv.2211.12136.

5 Richard T. Bumby. A problem with telephones. *SIAM. J. on Algebraic and Discrete Methods*, 2(1):13–18, 1981.

6 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.

7 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.

8 Kenneth L. Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

9 Pierluigi Crescenzi, Clémence Magnien, and Andrea Marino. Approximating the temporal neighbourhood function of large temporal graphs. *Algorithms*, 12(10):211, 2019.

10 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, Lecture Notes in Computer Science, pages 43–54. Springer, 2013.

11 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *ACM Journal of Experimental Algorithmics*, 23:1.7:1–1.7:56, 2018.

12 Arthur B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.

13 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Netw. Analys. Mining*, 8(1):61:1–61:29, 2018.

14 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

15 Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. Timetable information: Models and algorithms. In *ATMOS*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2004.

16 Karl Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83(1):154–166, 1995.

17 Stefano Pallottino and Maria Grazia Scutellà. Shortest path algorithms in transportation models: classical and innovative aspects. Technical Report TR-97-06, University of Pisa, 1997.

18 Stefano Pallottino and Maria Grazia Scutellà. *Equilibrium and Advanced Transportation Modelling*, chapter Shortest path algorithms in transportation models: classical and innovative aspects, pages 245–281. Kluwer Academic Publishers, 1998.

19 Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM J. Exp. Algorithmics*, 5:12, 2000.

20 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path Problems in Temporal Graphs. *VLDB Endowment*, 7(9):721–732, 2014.

# Complexity of Motion Planning of Arbitrarily Many Robots: Gadgets, Petri Nets, and Counter Machines

**Joshua Ani**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Michael Coulombe**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Erik D. Demaine**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Yevhenii Diomidov**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Timothy Gomez**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Dylan Hendrickson**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Jayson Lynch**
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

## Abstract

We extend the motion-planning-through-gadgets framework to several new scenarios involving various numbers of robots/agents, and analyze the complexity of the resulting motion-planning problems. While past work considers just one robot or one robot per player, most of our models allow for one or more locations to *spawn* new robots in each time step, leading to arbitrarily many robots. In the 0-player context, where all motion is deterministically forced, we prove that deciding whether any robot ever reaches a specified location is undecidable, by representing a counter machine. In the 1-player context, where the player can choose how to move the robots, we prove equivalence to Petri nets, EXPSPACE-completeness for reaching a specified location, PSPACE-completeness for reconfiguration, and ACKERMANN-completeness for reconfiguration when robots can be destroyed in addition to spawned. Finally, we consider a variation on the standard 2-player context where, instead of one robot per player, we have one robot shared by the players, along with a ko rule to prevent immediately undoing the previous move. We prove this impartial 2-player game EXPTIME-complete.

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

**Keywords and phrases** Gadgets, robots, undecidability, Petri nets

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.5

## 1 Introduction

Intuitively, motion planning is harder with more agents/robots. This paper formalizes this intuition by studying the effects of varying the number of robots in a recent combinatorial model for combinatorial motion planning and the resulting computational complexity.

Specifically, the ***motion-planning-through-gadgets framework*** was introduced in 2018 [10] and has had significant study since [12, 3, 6, 5, 11, 4, 17, 14]. In the original one-player setting, the framework considers a single agent/robot traversing a dynamic network of "gadgets", where each gadget has finite state and a finite set of traversals that the robot

© Joshua Ani, Michael Coulombe, Erik D. Demaine, Yevhenii Diomidov, Timothy Gomez, Dylan Hendrickson, and Jayson Lynch;
licensed under Creative Commons License CC-BY 4.0



can make depending on the state, and each traversal potentially changes the state (and thus which future traversals are possible). The goal is for the robot to traverse from one specified location to another (***reachability***), or for the system of gadgets to reach a desired state (***reconfiguration***) [5]. Existing results characterize in many settings which gadgets (in many cases, one extremely simple gadget) result in NP-complete or PSPACE-complete motion-planning problems, and which gadgets are simple enough to admit polynomial-time motion planning. This framework has already proved useful for analyzing the computational complexity of motion-planning problems involving modular robots [1], swarm robots [7, 8], and chemical reaction networks [2]. These applications all involve naturally multi-agent systems, so it is natural to consider how the complexity of the gadgets framework changes with more than one robot.

**1-player with arbitrarily many robots.** In Section 4, we consider a generalization of this 1-player gadget model to an arbitrary number of robots, and the player can move any one robot at a time. By itself, this extension does not lead to additional computational complexity: such motion planning remains in PSPACE, or in NP if each gadget can be traversed a limited number of times. To see the true effect of an arbitrary number of robots, we add one or two additional features: a ***spawner*** gadget that can create new robots, and optionally a ***destroyer*** gadget that can remove robots. For reachability, only the spawning ability matters – it is equivalent to having one "source" location with infinitely many robots – and we show that the complexity of motion planning grows to EXPTIME-complete with a simple single gadget called the ***symmetric self-closing door*** (previously shown PSPACE-complete without spawners [3]). For reconfiguration, we show that motion planning with a spawner and symmetric self-closing door is just PSPACE-complete (just like without a spawner), but when we add a destroyer, the complexity jumps to ACKERMANN-complete (in particular, the running time is not elementary). These results follow from a general equivalence to ***Petri nets*** – a much older and well-studied model of dynamic systems – whose complexity has very recently been characterized [15, 9].

**0-player with arbitrarily many robots.** In Section 3, we consider the same concepts in a 0-player setting, where every robot has a forced traversal during its turn, and spawners and robots take turns in a round-robin schedule. 0-player motion planning in the gadget framework with one robot was considered previously [6, 11], with the complexity naturally maxing out at PSPACE-completeness. With spawners and a handful of simple gadgets, we prove that the computational complexity of motion planning increases all the way to RE-completeness. In particular, the reachability problem becomes undecidable. This is a surprising contrast to the 1-player setting described above, where the problem is decidable.

**Impartial 2-player with a shared robot.** In Section 5, we consider changing the number of robots in the downward direction. Past study of 2-player motion planning in the gadget framework [12] considers one robot per player, with each player controlling their own robot. What happens if there is instead only one robot, shared by the two players? This variant results in an ***impartial*** game where the possible moves in a given state are the same no matter which player moves next. To prevent one player from always undoing the other player's moves, we introduce a ***ko rule***, which makes it illegal to perform two consecutive transitions in the same gadget. In this model, we show that 2-player motion planning is EXPTIME-complete for a broad family of gadgets called "reversible deterministic interacting $k$-tunnel gadget", matching a previous result for 2-player motion planning with one robot per player [12]. In other words, reducing the number of robots in this way does not affect the complexity of the problem (at least for the gadgets understood so far).

## 2 Standard Gadget Model

We now define the gadget model of motion planning, introduced in [10].

In general, a ***gadget*** consists of a finite number of ***locations*** (entrances/exits) and a finite number of ***states***. Each state $S$ of the gadget defines a labeled directed graph on the locations, where a directed edge $(a, b)$ with label $S'$ means that a robot can enter the gadget at location $a$ and exit at location $b$, changing the state of the gadget from $S$ to $S'$. Equivalently, a gadget is specified by its ***transition graph***, a directed graph whose vertices are state/location pairs, where a directed edge from $(S, a)$ to $(S', b)$ represents that the robot can traverse the gadget from $a$ to $b$ if it is in state $S$, and that such traversal will change the gadget's state to $S'$. Gadgets are ***local*** in the sense that traversing a gadget does not change the state of any other gadgets.

A ***system of gadgets*** consists of gadgets, their initial states, and a ***connection graph*** on the gadgets' locations. If two locations $a$ and $b$ of two gadgets (possibly the same gadget) are connected by a path in the connection graph, then a robot can traverse freely between $a$ and $b$ (outside the gadgets). (Equivalently, we can think of locations $a$ and $b$ as being identified, effectively contracting connected components of the connection graph.) These are all the ways that the robot can move: exterior to gadgets using the connection graph, and traversing gadgets according to their current states.

Previous work has focused on the robot reachability[1] problem [10, 12]:

▶ **Definition 2.1.** *For a gadget $G$, **robot reachability for $G$** is the following decision problem. Given a system of gadgets consisting of copies of $G$, the starting location(s), and a win location, is there a path a robot can take from the starting location to the win location?*

Gadget reconfiguration, which had target states for the gadgets to be in, was considered in [5] and [14]. We additionally investigate a problem where we have target states and multiple locations which require specific numbers of robots.

▶ **Definition 2.2.** *For a gadget $G$, the **multi-robot targeted reconfiguration problem for $G$** is the following decision problem. Given a system of gadgets consisting of copies of $G$, the starting location(s), and a target configuration of gadgets and robots, is there a sequence of moves the robots can take to reach the target configuration?*

[12] also defines 2-player and team analogues of this problem. In this case, each player has their own starting and win locations, and the players take turns making a single transition across a gadget (and any movement in the connection graph). The winner is the player who reaches their win location first. The decision problem is whether a particular player or team can force a win. When there are multiple robots, we are asking whether any of them can reach the win location.

We will consider several specific classes of gadgets.

▶ **Definition 2.3.** *A **$k$-tunnel** gadget has $2k$ locations, which are partitioned into $k$ pairs called **tunnels**, such that every transition is between two locations in the same tunnel.*

Most of the gadgets we consider are $k$-tunnel.

[1] In [10, 12], "reachability" refers to whether an agent/robot can reach a target location. Here we refer to it as *robot reachability* since for models such as Petri-nets the Reachability problem refers to whether a full configuration is reachable.

▶ **Definition 2.4.** *The **state-transition graph** of a gadget is the directed graph which has a vertex for each state, and an edge $S \to S'$ for each transition from state $S$ to $S'$. A **DAG** gadget is a gadget whose state-transition graph is acyclic.*

DAG gadgets naturally lead to bounded problems, since they can be traversed a bounded number of times. The complexity of the reachability problem for DAG $k$-tunnel gadgets, as well as the 2-player and team games, is characterized in [12].

▶ **Definition 2.5.** *A gadget is **deterministic** if every traversal can put it in only one state and every location has at most 1 traversal from it. More precisely, its transition graph has maximum out-degree 1.*

▶ **Definition 2.6.** *A gadget is **reversible** if every transition can be reversed. More precisely, its transition graph is undirected.*

Reversible deterministic gadgets are gadgets whose transition graphs are partial matchings, and they naturally lead to unbounded problems. [12] characterizes the complexity of reachability for reversible deterministic $k$-tunnel gadgets and partially characterizes the complexity of the 2-player and team games.

We define the decision problems we consider in their corresponding sections.

# 3 0-Player Motion Planning with Spawners

In this section, we describe a model of 0-player motion planning, introduce the spawner gadget, and show that 0-player motion planning with spawners is RE-complete, implying undecidability. RE-completeness is defined in terms of arbitrary computable many-one reductions; in particular, they don't have to run in polynomial time. We will use the fact that the halting problem for 3-counter machines is RE-complete [18].

## 3.1 Model

In ***0-player directed-edge motion planning*** (with one robot), we modify 1-player motion planning by removing the player's ability to control the robot, and specifying directions on the connections between gadget locations. More precisely, the connection graph is now a directed graph such that each gadget location has only incoming edges (meaning that the robot enters the gadget from that location), or only outgoing edges and at most one such edge (meaning that the robot exits the gadget from that location); and all gadgets must be deterministic.[2] Thus the robot moves on its own, moving in the direction of the edge it is on and traversing any gadgets it encounters. The reachability question asks whether the robot reaches a specified target location in finite time.

Because the state of this system can be encoded in a polynomial number of bits (the state for each gadget and the location of the robot), this reachability problem is in PSPACE as in other 0-player models of the gadget framework [6, 11].

Our extension is to define the ***spawner*** gadget: a 1-location gadget that spawns a new robot in each round, appearing at its only location. We now define 0-player directed-edge motion planning to take into account multiple robots and spawners. ***0-player directed-edge***

[2] There was no need to apply directions to the connection graph in [6] because each location acted exclusively as either the start of transitions or the end of transitions. In [11] the connections were undirected and it was assumed the robot proceeded away from the gadget where it just traversed.

***motion planning with spawners*** is divided into rounds. In each round, each robot takes a turn in spawn order, and then each spawner spawns a robot (in a predefined spawning order). A robot's turn consists of it moving along the directed edge it is on until it either traverses a gadget or it gets stuck (i.e., reaches a point where all edges are directed to its position). The reachability question asks whether any robot reaches a specified target location in finite time.

▶ **Lemma 3.1.** *Deciding robot reachability in 0-player directed-edge motion planning with spawners with any set of gadgets is in RE.*

**Proof.** After each step of the game, there will still be a finite, if increasing, number of robots. Thus to confirm if at least 1 robot can reach the win location in finite time we can simply simulate the game for the needed finite number of steps. ◀

## 3.2 RE-hardness

We show that deciding robot reachability in 0-player directed-edge motion planning with spawners is RE-hard by reduction from the halting problem by simulating a 3-counter machine. First we introduce the gadgets that we show RE-hard.

**Increment gadget.** The ***increment gadget*** is a 4-state 10-location gadget containing a 3-path ***lock branch*** and a 3-path ***path selector*** (Figure 1). When a robot traverses a path in the path selector, it enables a single path in the lock branch and locks the path selector. When a robot traverses a path in the lock branch, the gadget reverts to the original state.

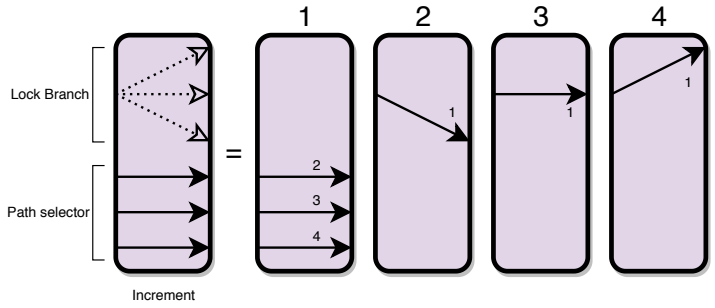



**Figure 1** The increment gadget, shown with state transitions.

**Register gadget.** The ***register gadget*** is a 3-state 10-location gadget containing a ***path selector***, a ***processing branch***, and a ***response branch*** (Figure 2). When a robot traverses the top path selector path, the path selector is locked and a path in the processing branch is enabled. When a robot traverses the bottom path selector path, the path selector is locked and the other processing branch path and a path in the response branch are enabled. If a robot traverses any non-path-selector path, the gadget reverts to the original state.

**UPDSDS gadget.** For the following theorem, we will also use the ***UPDSDS*** gadget. This gadget has two states "up" and "down", a tunnel which sets the state to "up", and two ***set-up switches*** which each have one input and two outputs, where the output taken depends on the state and traversing the switch sets the state to "down"'.

▶ **Theorem 3.2.** *Deciding robot reachability for 0-player directed-edge motion planning with spawners is RE-hard with the spawner, increment, register, and UPDSDS gadgets combined.*

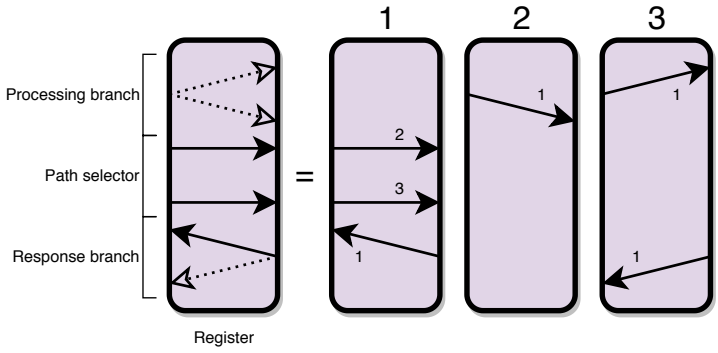



**Figure 2** The register gadget, shown with state transitions.

**Proof.** We reduce from the halting problem of the 3-counter machine with `INC(r)`, `DEC(r)`, and `JZ(r, z)` instructions, which is undecidable ([18]). We will need to implement the `INC(r)` (increment register $r$ by 1), `DEC(r)` (decrement $r$ by 1), and `JZ(r, z)` (jump to instruction $z$ if $r$ is 0) instructions of a counter machine. We will not worry about decrementing a register that is already 0, because all `DEC` instructions can be preceded by `JZ` to guard against that. We will also implement the `HALT` instruction, which should result in a win.

First we implement a ***register***, which will store a nonnegative integer, just like a register in a counter machine. This, of course, uses the register gadget, and the implementation is shown in Figure 3. In this implementation, the value of a register gadget is the number of robots stuck at the entrance of the processing branch. If a robot $b$ crosses the *decrement in* path, a single robot can cross the gadget to the sink, where it is stuck forever, and all other robots stuck at the entrance stay stuck. Robot $b$ goes through the *out* path on its next turn. This decrements the value of the gadget by 1, thus implementing `DEC`, taking 1 round to process. If a robot $b$ crosses the *jump-zero in* path, then if the gadget's value is nonzero, a single robot $b'$ crosses the top path of the processing branch, reverting the gadget's state, and forcing $b$ to traverse the top path of the response branch on its next turn, which leads to the *out* path. $b'$ gets stuck back at the entrance on its next turn. However, if the gadget's value is 0, then no robot will traverse the processing branch, which lets $b$ traverse the bottom path of the response branch on its next turn. This does not change the value of the gadget, and changes the path of $b$ iff the value is 0, thus implementing `JZ`, taking 2 rounds to process.

To implement `INC`, we need a place that robots can come from. For this, we have the setup shown in Figure 4. This setup contains a spawner gadget. Spawned robots go through the US gadget (a set-up switch, simulated by using one switch of the UPDSDS gadget and flipping it) to the entrance of the lock branch of the increment gadget and get stuck. It takes 2 turns for this to happen. The first robot $b$ to get spawned instead takes the bottom path of the US gadget and executes the program. So during the 4th and later rounds, an extra robot gets stuck at the increment gadget. When robot $b$ goes through the *increment* $r_i$ *in* path, a single robot $b'$ at the increment gadget traverses the lock branch, goes to the *income* entrance of $r_i$, and gets stuck at that register gadget's processing branch on its next turn, incrementing said register gadget's value. In the process, the increment gadget reverts to its original state. This implements `INC`, taking 2 rounds to process, and we only need to make sure that $b$ does not traverse the path selector of the increment gadget before the 4th round to ensure that there will be a robot $b'$ that goes to a register.

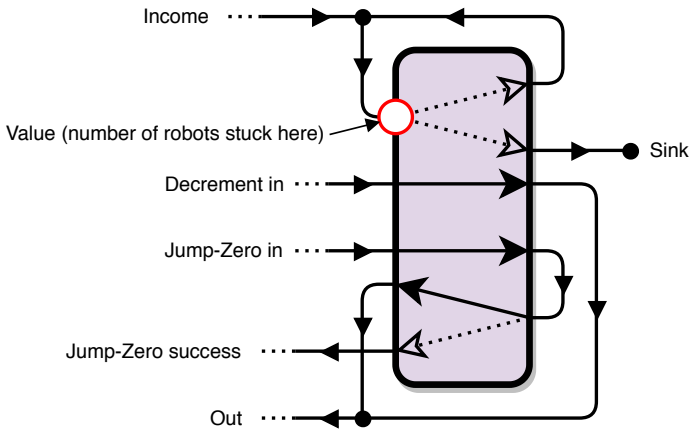



**Figure 3** Implementation of the register of a counter machine.

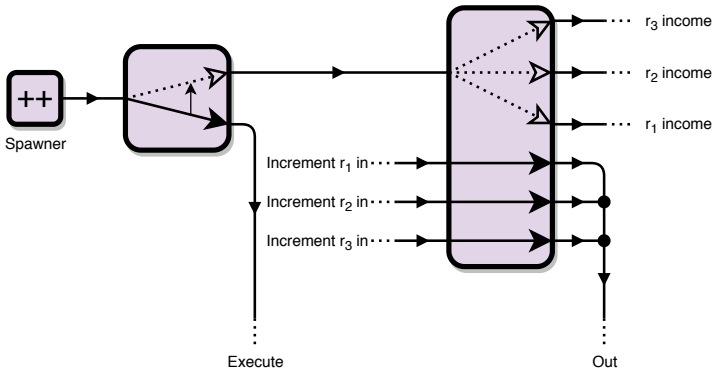



**Figure 4** The context of the increment gadget, along with the spawner and a US gadget.

We also need to implement the program, and we use UPDSDS gadgets for that, as shown in Figure 5. A UPDSDS-gadget instruction contains an *execute in* entrance, a *pass in* entrance, a *jump in* entrance, a *jump destination* entrance, an *execute out* exit, an *execute next* exit, a *pass next* exit, a *jump next* exit, and a *jump out* exit. Only the executor robot is allowed to traverse this gadget.

The *execute out* exit leads to the proper location of the increment or register gadgets. For an `INC(r)` instruction, it leads to the *increment r in* entrance of the increment gadget. For a `DEC(r)` instruction, it leads to the *decrement in* entrance of the register gadget for register $r$. For a `JZ(r, z)` instruction, it leads to the *jump-zero in* entrance of the register gadget for register $r$. For a `HALT` instruction, it leads directly to the win location.

The *execute next* exit leads to the *execute in* entrance of the next instruction. The *pass next* exit leads to the *pass in* entrance of the next instruction. The *jump out* exit leads to the *jump destination* entrance of instruction $z$ for a `JZ(r, z)` gadget, and doesn't exist otherwise. The *jump next* exit leads to the *jump in* entrance of the next instruction.

This reduction can be done in polynomial time with respect to the number of instructions, because each instruction is simulated with 1 UPDSDS gadget, and there are a constant number of constant-size gadgets other than these.

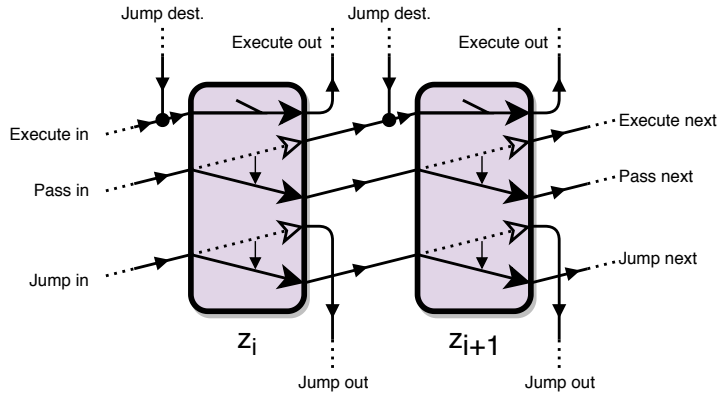



**Figure 5** Two instructions implemented using UPDSDS gadgets.

We now describe the behavior of the entire simulation, with an example shown in Figure 6.

- A robot spawns from the spawner.
- The robot that spawned takes the bottom path of the US gadget, setting it to the *up* state permanently. This robot is the executor robot. Another robot spawns from the spawner.
- The executor robot takes the top path of the UPDSDS gadget representing the first instruction. The newly spawned robot crosses the US gadget. Another robot spawns from the spawner.
- If the executor robot is executing an `INC` instruction, it traverses the path selector of the increment gadget. This is the 4th (or later) round, so there will be a robot ready to traverse the lock branch of the increment gadget.
- When the executor robot finishes executing an instruction that doesn't lead to a jump, it travels along the upper set-down switches of the UPDSDS gadgets until it finds the one representing the instruction it was executing. It resets that gadget and executes the next instruction, flipping the state of the next UPDSDS gadget.
- If the instruction led to a jump instead, the executor robot travels along the lower set-down switches of the UPDSDS gadgets until it finds the one representing the instruction it was executing. It resets that gadget and takes the *jump next* path to the destination UPDSDS gadget of the jump, then executes the corresponding instruction.
- If the executor robot reaches the top path of the UPDSDS gadget representing the `HALT` instruction, it goes to the win location.

So this simulates a 3-counter machine. So if the 3-counter machine halts, then a robot will reach the win location in finite time, and vice versa. ◀

## 4 1-Player Motion Planning with Spawners and/or Destroyers

In this section, we investigate 1-player motion planning with multiple robots, where a single player controls a set of robots, with the ability to separately command each, moving any one robot at a time. There is no limit to the number of robots that can be at a given location. We include a ***spawner*** gadget (as in Section 3) which the player can use to produce a new robot at a specific location, providing an unlimited source of robots at that location. We optionally also include a ***destroyer*** gadget, which deletes any robot that reaches a specified sink location; such removal plays a role when we consider the ***targeted reconfiguration***

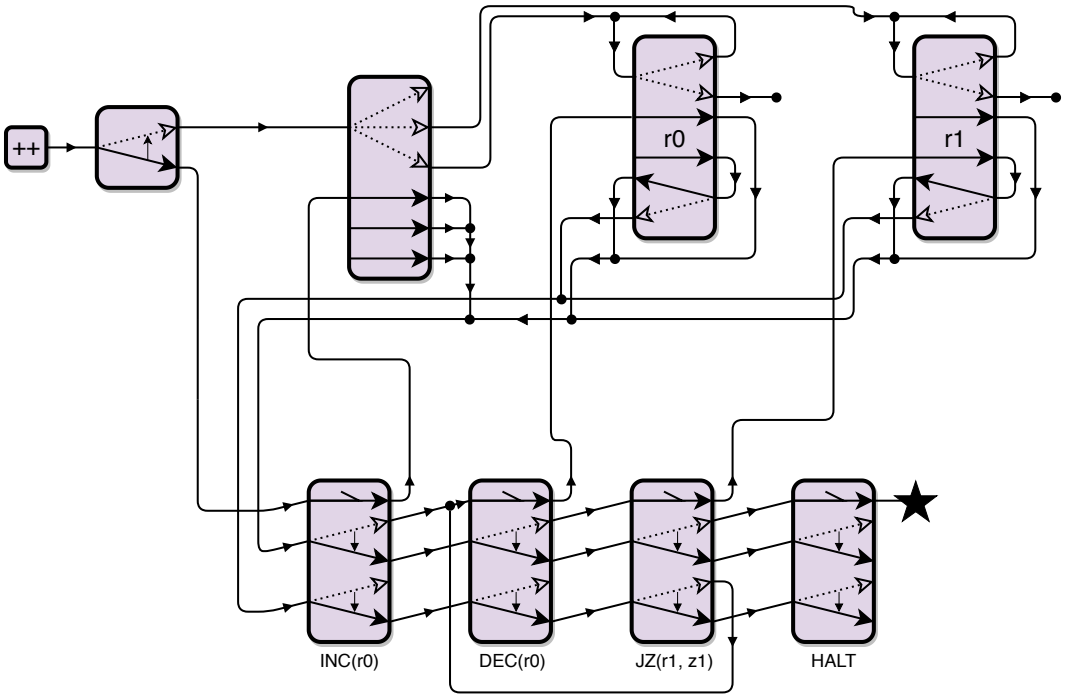



**Figure 6** A 2-counter machine constructed with the gadgets. 2 counters are shown instead of 3 to save space.

problem where the goal is to achieve an exact pattern of robots at the locations. If a system of gadgets only has a single spawner gadget we call that gadget the ***source*** and if the system only has a single destroyer gadget we call that the ***sink***.

We show an equivalence between this 1-player motion planning problem and corresponding problems on Petri nets. Through these connections, we establish EXPSPACE-completeness for reachability; PSPACE-completeness for reconfiguration with a spawner; and ACKERMANN-completeness for reconfiguration with a spawner and a destroyer.

## 4.1 Petri Nets

Petri nets are used to model distributed systems using tokens divided into dishes, and rules which define possible interactions between dishes. This is a natural model since many equivalent models have been defined such as Vector Addition Systems and Chemical Reaction Networks.

▶ **Definition 4.1.** *A **Petri net** $\{D, R\}$ consists of a set of dishes $D$ and rules $R$. A configuration $t$ is a vector over the elements of $D$ which represents the number of tokens in each dish. Each rule $(u, v) \in R$ is a pair of vectors over $D$. A rule can be applied to a configuration $d_0$ if $d_0 - u$ contains no negative integers to change the configuration to $d_1 = d_0 - u + v$. The volume of a configuration denoted $|d|$ is the sum of all its elements.*

▶ **Definition 4.2.** *A reachable set for a Petri-net configuration, denoted $REACH_P(\{D, R\}, t)$, is the set of configurations of a Petri net reachable starting in configuration $t$ and applying rules from $R$.*

We can view a system of gadgets with multiple robots as a set of gadget states $\Gamma$ and a vector $l$ indicating the counts of robots at each location. We can define the set of reachable targeted configurations as $REACH(\Gamma, l)$ similarity to Petri nets.

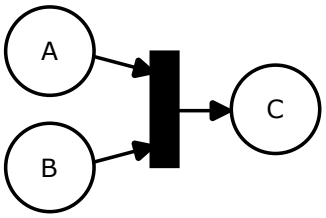



**Figure 7** General Petri-net rule $(u, v)$, where $u$'s nonzero dishes are shown on the left side and $v$'s nonzero dishes are shown on the right side.

## 4.2 Equivalence between Petri Nets and Gadgets

We present transformations that turn Petri nets into gadgets, and gadgets into Petri nets. We use these simulations to prove the complexity of robot reachability and reconfiguration with arbitrarily many robots.

**Gadgets to Petri Nets.** We can transform a set of gadgets into a Petri net where each location, besides the source and sink, is represented as a *robot dish*. Each gadget besides the spawner and destroyer is given a number of *state dishes* equal to its states, and each transition of the gadget is represented by a *rule*. The set of dishes $D$ is $D_{STATE} \cup D_{LOCT}$, the union of state and robot dish sets, respectively.

A configuration of robots and gadgets is represented by a Petri-net configuration $t$ satisfying the following:

- Each $k$-state gadget is simulated by $k$ unique dishes in $D_{STATE}$, one per state. The state of the gadget is represented by a single token which is contained in the corresponding dish, and the other $k - 1$ dishes are empty.
- Each location in the system of gadgets is simulated by a unique dish in $D_{LOCT}$. The number of tokens in that dish is equal to the number of robots at that location.

A Petri net $\{D, R\}$ simulates a system of gadgets $G$ if for any configuration $\{\Gamma, l\}$ of $G$ represented by Petri-net configuration $t$, each configuration in $REACH_G(\Gamma, I)$ is represented by a configuration $REACH_P(\{D, R\}, t)$ and each configuration in $REACH_P(\{D, R\}, t)$ represents a configuration in $REACH_G(\Gamma, I)$.

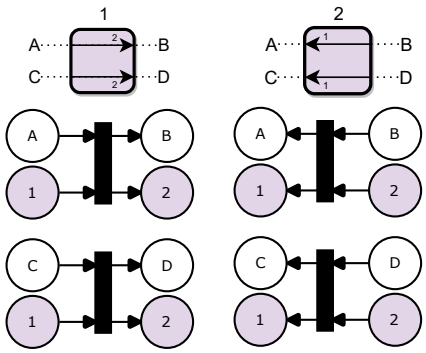



**Figure 8** Petri-net rules which simulate a 2-tunnel toggle gadget.

▶ **Lemma 4.3.** *For any set of deterministic gadgets $S$, any system of multiple copies of gadgets in $S$ with a spawner (and optionally, a destroyer) can be simulated by a Petri net.*

**Proof.** We first explain how to create the rules for gadgets that are not connected to the source or sink locations. Each gadget transition will be represented by a unique rule. For example the 2-tunnel toggle gadget is shown in Figure 8 and has four transitions. It can be traversed:

- from $A$ to $B$ in state 1,
- from $C$ to $D$ in state 1,
- from $B$ to $A$ in state 2, and
- from $D$ to $C$ in state 2.

The four corresponding rules for the gadget are drawn in Figure 8 as well. Each rule takes in one token from a robot dish and one from a state dish, and places one token in a robot dish and one in a state dish. The token being moved between robot dishes models moving one robot across a gadget, and the token being moved between state dishes models the state change of the gadget.

If a gadget is connected to the source, any transition from the source is represented by a rule that only takes in a state token, producing two tokens. One token is output to a location dish and one to a state dish. If a transition is connected to the sink then the rule takes in two tokens and outputs only a state token. These special cases are shown in Figure 9. Note that we do not have an actual dish for the source so the player may spawn multiple robots at the source but they do not appear in the simulation until they traverse a gadget.

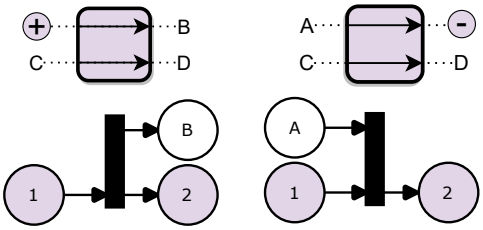



**Figure 9** Left: Rule we include when a gadget can be traversed from the source. Right: Rule we include when a traversal leads to the sink.

For each configuration of a system of gadgets, there exists a configuration of the Petri net with dishes that represent the gadgets and locations. Each rule of the Petri net acts as a traversal of a robot changing the state of a gadget. The rules need the gadgets state token to be in the correct dish, and a robot token in the location dish representing the start traversal. ◀

**Petri Nets to Gadgets.** We simulate a Petri net with symmetric self-closing doors using a location for each dish, where each rule is represented by multiple gadgets. We also have a single ***control robot*** which starts in a location we call the ***control room***. The other robots are ***token robots*** which represent the tokens in each dish. At a high level, our simulation works by "consuming" the input tokens to a rule to open a series of tunnels for the control robot to traverse. The control robot then opens a gadget for each output to allow token robots to traverse into their new dishes. We use the source and sink to increase and decrease rules as needed. Figure 11 gives an overview.

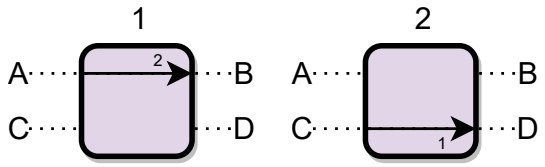



**Figure 10** Symmetric self-closing door.

**Symmetric self-closing door.** The ***symmetric self-closing door*** is a 2-state 2-tunnel gadget shown in Figure 10. The states are $\{1, 2\}$ and the traversals are

- in state 1 from $A$ to $B$ changing state to 2, and
- in state 2 from $C$ to $D$ changing state to 1.

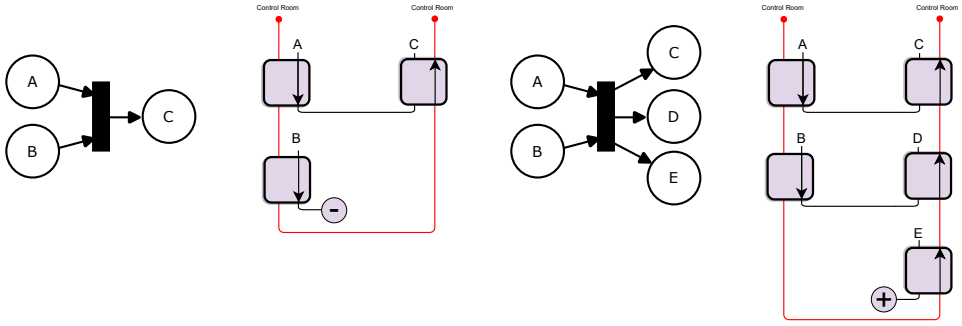



**Figure 11** How to simulate a rule which decreases volume (Left) and a rule which increases volume (Right).

Using this simulation we prove two problems in Petri-nets are polynomial time reducible to the gadgets problems we are interested in. [13] lists many problems including the ones we describe here[3]. First is production, this problem asks given a Petri-net configuration and a target dish, does there exist a reachable configuration which contains at least one token in the target dish. Configuration reachability asks given an initial and target configuration, is the target reachable from the initial configuration.

▶ **Lemma 4.4.** *Production in Petri nets is polynomial time reducible to robot reachability with the symmetric self-closing door and a spawner. Configuration reachability in Petri nets is polynomial-time reducible to multi-robot targeted reconfiguration with the symmetric self-closing door and a spawner.*

**Proof.** For a rule $(a, b)$ we include $|a| + |b|$ copies of the gadgets. There is a gadget for each input to the rule; these gadgets can be traversed from the location representing an input dish to an intermediate location, opening another tunnel for the control robot to traverse. The control robot must traverse all the input gadgets the goes through the tunnels of the output gadgets. The control robot opens the doors of these gadgets allowing the robots moving from an intermediate wire to traverse to a location representing the output dishes.

If a rule would increase the volume, the surplus output gadgets will allow traversal from the spawn location instead of an input gadget. If a rule decreases the volume, then the surplus input gadgets send robots to a "sink" location instead of an output gadget. We do

[3] Problems names may differ.

not require a true sink in this case because we can add an extra location which robots can be held instead of being deleted. If we do not connect this location to any other gadget, then the robots can never leave and can be thought of as having left the system.

Production reduces to robot reachability since a robot can reach a location if and only if a token can reach the corresponding dish. If token is placed in a dish, it must have moved through a rule gadget. The robot can only move through a rule gadget if the number of robots in the dishes are at least the number of tokens of the left hand side of the rules to open the tunnels for the control robot to move through.

Configuration reachability in Petri nets reduces to multi-robot targeted reconfiguration. The target and initial states of the gadgets are the same. The only difference between the initial configuration and the target is the number of robots at each location, equal to the counts in the instance of Configuration reachability for Petri nets. The number of robots at each location is equal to the number of tokens in each dish. The targets for each intermediate wire is 0 and in the control room 1. Thus, it is never beneficial to partially traverse a rule gadget. ◀

## 4.3 Complexity of Reachability

The reachability problem for a single robot is very similar to the well-studied problem in Petri nets called coverage. The input to the coverage problem is a Petri net and a vector of required token amounts in each dish, and the output is yes if and only if there exists a rule application sequence to reach a configuration with at least the required number of tokens in each dish.

▶ **Definition 4.5** (Coverage Problem). ***Input:*** *A Petri net* $\{D, R\}$*, and vectors* $d_0$ *and* $d_c$*.*

***Output:*** *Does there exist a reachable configuration* $d \in REACH(\{D, R\}, d_0)$ *such that* $d[k] \geq d_c[k]$ *for all* $0 \leq k < |D|$*.*

▶ **Theorem 4.6.** *Robot reachability is EXPSPACE-complete with symmetric self-closing doors, a spawner, and optionally a destroyer.*

**Proof.** We can solve robot reachability by converting the system of gadgets to a Petri net which simulates it as in Lemma 4.3. In this simulation, a token can be placed in a location dish if and only if a robot can reach that location represented by that dish. Determining if a single token can be placed in a target dish, the production problem, is a special case of coverage problem where the target dish is labeled with 1 and all others labeled with 0. We can use the exponential-space algorithm for Petri-net coverage shown in [19] to solve robot reachability. When simulating the sink we require rules that decrease the volume of a Petri net. This algorithm works for general Petri nets so it implies membership with a sink.

For hardness, we first reduce Petri-net coverage to Petri-net production by adding a target dish $T$ starting with 0 tokens and a new rule. This rule takes as input the number of tokens equal to the goal of the coverage problem and produces one token to the $t$ dish. This token can only produced if the reach a configuration that has at least the target number of each species. We then use Lemma 4.4 to reduce production to robot reachability with the self-closing symmetric door and a spawner. It is relevant to note the first reduction does not work when exactly the target numbers are required. The reduction works even when not allowing the sink as described in Lemma 4.4. ◀

## 4.4 Complexity of Reconfiguration

The reconfiguration problem has been studied in the single-robot case as the problem of moving the robot through the system of gadgets so that each gadget is in a desired final state. Targeted reconfiguration not only asked about the final states of the gadgets, but the location of the robot as well. Here, we study multi-robot targeted reconfiguration which requires both that all gadgets are in specified final states and that each location contains a target number of robots.

▶ **Definition 4.7.** *For a gadget $G$, the **multi-robot targeted reconfiguration problem for $G$** is the following decision problem. Given a system of gadgets consisting of copies of $G$ and the starting location(s) a target configuration of gadgets and robots, is there a sequence of moves the robots can take to reach the target configuration?*

The complexity of multi-robot targeted reconfiguration depends on whether we allow a destroyer. If we do not allow for a destroyer, the complexity is bounded by polynomial space since we can never have more robots than the total target size. If we allow for the ability to destroy robots, then the reconfiguration problem is the same as the configuration reachability problem in Petri nets from our relations between the models above. This is a fundamental problem about Petri nets and was only recently shown to be ACKERMANN-complete [15, 9].

▶ **Theorem 4.8.** *Multi-robot targeted reconfiguration is ACKERMANN-complete with symmetric self-closing doors, a spawner, and a destroyer.*

**Proof.** For membership we can solve multi-robot target reconfiguration by converting the gadgets to the Petri net using Lemma 4.3. The target configuration is a state token for each gadget in the dish of its target state, and a number of tokens in each location dish as the number of robots in the target configuration. We can then call the ACKERMANN algorithm for configuration reachability in Petri nets shown in [16].

For hardness we can reduce from configuration reachability. It was shown in [9] that configuration reachability is ACKERMANN-hard. ◀

The reduction presented in [9] vitally uses the ability of Petri nets to delete tokens, so we must use a sink in our simulation. Without a sink, we have PSPACE-completeness for multi-robot targeted reconfiguration.

▶ **Theorem 4.9.** *Multi-robot targeted reconfiguration for symmetric self-closing doors and a spawner is PSPACE-complete.*

**Proof.** Consider the input to the reconfiguration problem: two configurations of a system of gadgets. Namely, the start and end state of all the gadgets, and a start and end integer for each location. Since we can never destroy a robot once it is spawned, it always exists, so the player cannot spawn more robots than the total number of robots in the target configuration. We can then solve this problem in NPSPACE by nondeterministically selecting a robot to move, either from the source or another location. If we ever increase the total number of robots above the target we may reject. If we ever reach the configuration with the correct gadget states and robots at each location accept. Since PSPACE = NPSPACE we get membership.

We inherit hardness from the 1-player single-robot case by not including the source or connecting it to an unreachable location. ◀

# 5 Impartial Unbounded 2-Player Motion Planning

In this section, we describe the 2-player impartial motion planning game and show that it is EXPTIME-complete for any reversible deterministic gadget.

## 5.1 Model

In the ***2-player impartial motion planning game***, two players control the same robot in a system of gadgets. Player 1 moves first, then Player 2 moves, then play repeats. On a given player's turn, they move the robot arbitrarily along the connection graph and through exactly one transition of a gadget. There is also a ***ko rule***: The robot cannot traverse the same gadget on a player's turn as it traversed on their opponent's previous turn. If a player cannot make the robot traverse a gadget without breaking the ko rule, that player loses and the other player wins.

▶ **Lemma 5.1.** *Deciding whether Player 1 has a deterministic winning strategy in the 2-player impartial motion planning game is in EXPTIME for any set of gadgets.*

**Proof.** An alternating Turing machine can solve the problem by using existential states to guess Player 1's moves and universal states to guess Player 2's moves, accepting when Player 1 wins and rejecting when Player 2 wins. This takes only polynomial space because the configuration of the game can be described in polynomial space. The machine can reject after a number of turns at least the number of configurations, which is at most exponential and thus can be counted to in polynomial space. Hence the problem is in APSPACE = EXPTIME. ◀

## 5.2 Hardness

We introduce the ***locking 2-toggle***, introduced in [12] and shown in Figure 12. States 1 and 3 are ***leaf states*** and state 2 is the ***nonleaf state***. If a robot crosses a tunnel in state 2, the tunnel flips direction and the other tunnel locks. Crossing a tunnel again will reverse this effect.

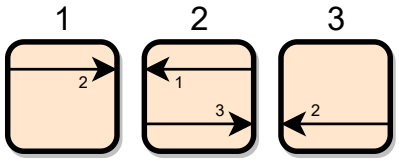



**Figure 12** The locking 2-toggle.

▶ **Theorem 5.2.** *Deciding whether Player 1 has a deterministic winning strategy in the 2-player impartial motion planning game is EXPTIME-hard for the locking 2-toggle.*

**Proof.** We reduce from $G_4$ as defined in [20]. $G_4$ is a 2-player game involving Boolean variables where the players flip a variable on their turn and try to be the one to satisfy a common DNF Boolean formula with 13 variables per clause (a 13-DNF). Players have their own variables and can't flip their opponent's variables, and a player may flip 1 variable on their turn or pass their turn. There is no ko rule.

We start the robot next to a 1-toggle (a single tunnel of a locking 2-toggle) as shown in Figure 13. This 1-toggle is called the ***alternator***. On each side of the alternator is a variable system for each player, which consists of variable branching and variable setting loops. The

***variable branching***, as shown in Figure 14, has 2 locking 2-toggles before each branch. These start in the nonleaf state. At the end of each path is a ***variable flipping loop***, which is shown in 15. The variable flipping loop for variable $v$ contains 2 locking 2-toggles per instance of $v$ or $\neg v$ in the 13-DNF formula of the $G_4$ instance, as well as an path to the 13-DNF checker with 2 1-toggles on it. The locking 2-toggles representing $v$ start in the nonleaf state iff $v$ starts True in $G_4$, and the locking 2-toggles representing $\neg x$ start in the leaf state iff $x$ starts True in $G_4$. One path of the variable branch, on the other hand, leads to a ***pass loop***, which is a variable flipping loop with 2 1-toggles in the loop instead of the locking 2-toggles. The 13-DNF checker contains a path for each clause in the 13-DNF, and each path contains a locking 2-toggle representing $v$, the same as one of the locking 2-toggles representing $v$ in the variable flipping loop of $v$, followed by a 1-toggle, for each variable $v$ in the corresponding clause. The paths all lead to a final 1-toggle called the ***finish line***. This reduction can be done in polynomial time, as each variable and clause in $G_4$ is converted to a polynomial number of constant-size gadgets.

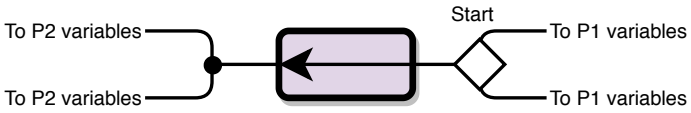



**Figure 13** The robot's starting position, and the 1-toggle that's called the *alternator*.

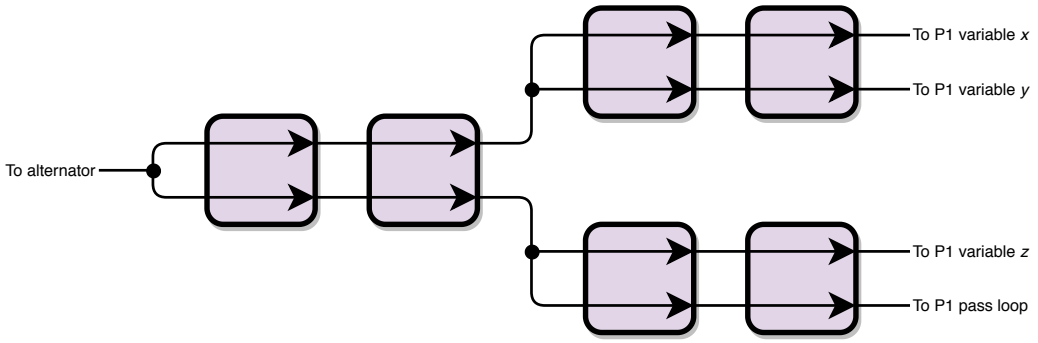



**Figure 14** The variable branching for Player 1. Player 2's variable branching is on the other side of the alternator. In this example, player 1 has 3 variables: $x$, $y$, and $z$.

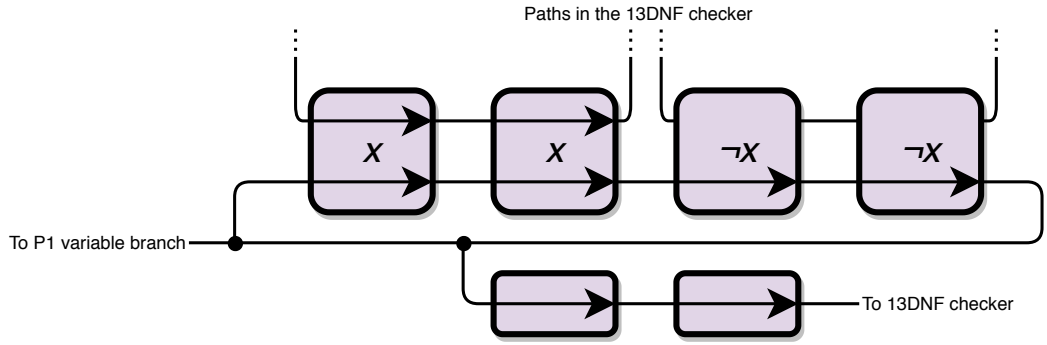



**Figure 15** The variable flipping loop for variable $x$. This example represents the case where the 13-DNF has 1 instance of $x$ and 1 instance of $\neg x$. Currently, $x$ is True.

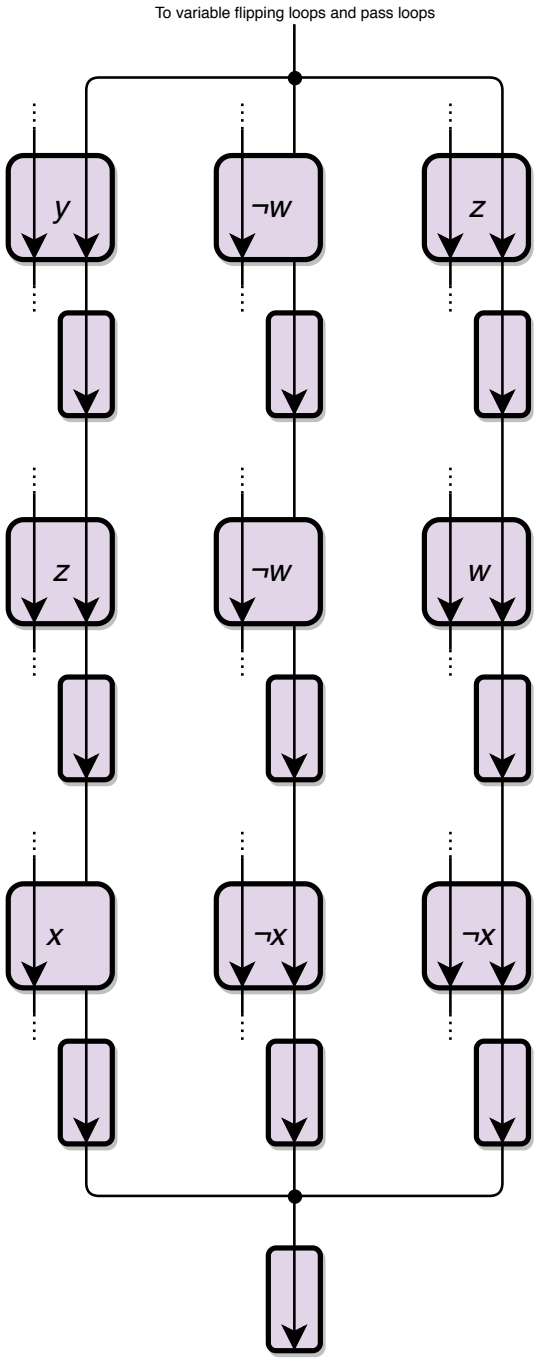



**Figure 16** A 13-DNF checker, except that it represents a 3-DNF. This example represents $(y \vee z \vee x) \wedge (\neg w \vee \neg w \vee \neg x) \wedge (z \vee w \vee \neg x)$. The dotted paths are part of variable setting loops.

During intended play:

- Player 1 moves the robot through variable branching to select a variable to set. Because the locking 2-toggles are doubled, and because of the ko rule, Player 2 has no choice but to second Player 1's choices. Player 1 could also move the robot to the pass loop.

- Player 1 moves the robot around a variable selection loop, a variable by flipping whether each locking 2-toggle is locked or not. If they're in the pass loop, they just go around the loop. Again, Player 2 has no choice since the number of gadgets in the path is even.
- Player 1 either moves the robot to the 13-DNF checker or back through the variable branching to the alternator.
- If Player 1 moves it back, they make it cross the alternator, and Player 2 goes through the same steps, but on the other side of the alternator.
- If a player moves the robot to the 13-DNF checker, they pick a path. If that path's corresponding clause in the 13-DNF is currently satisfied, they cross the finish line and win, since their opponent then has no legal moves. Otherwise, they get blocked by the first variable set to False, making their opponent win.

So Player 1 has the initiative and takes a $G_4$ turn on one side of the alternator, and Player 2 has the initiative and takes a $G_4$ turn on the other side. It is correct for a player to move the robot to the 13-DNF checker iff the 13-DNF is currently satisfied.

We will now look at ways that the players can try to break the simulation of $G_4$:

- Player 1 can make the robot cross the alternator as their first move. However, this lets Player 2 flip a variable or pass first. If Player 1 can win this way, they can also win by passing (moving the robot around the pass loop) first and then giving the initiative to Player 2. So not crossing the alternator first is always a correct move.
- A player can move the robot to a variable flipping loop and cut to the 13-DNF checker. However, if the player can win this way, they can win by passing and moving the robot to the 13-DNF checker.
- A player can try to turn around and flip another variable on the way back to the alternator. However, the ko rule prevents this.
- A player can try to move the robot to some other variable flipping loop from the start of the 13-DNF checker. However, 1-toggles will block the way.

Thus, the players are effectively forced to play $G_4$ in this game. Therefore, if Player 1 has a deterministic winning strategy in the $G_4$ instance, then they have one in this game, and if Player 1 has a deterministic winning strategy in this game, then they have one in the $G_4$ instance as well. ◀

▶ **Theorem 5.3.** *Deciding whether Player 1 has a deterministic winning strategy in the 2-player impartial motion planning game is EXPTIME-hard for* any interacting $k$-tunnel reversible deterministic gadget.

**Proof.** Figure 17 shows two tunnels that any interacting $k$-tunnel reversible deterministic gadget must have, as proved in [12, Section 2.1], which further shows that these tunnels can be used to simulate a locking 2-toggle. For 2-player impartial motion planning, however, we must be careful of the simulation. To preserve parity, each traversal in the locking 2-toggle must correspond to an odd number of traversals in the simulation. In addition, if a traversal is not allowed, it must be blocked after an even number of traversals so the player who started moving the robot along that path loses. And to simulate the gadget ko rule, the gadgets at the ends of the simulation must be in the way of both paths. If all the constraints are met, then if a player makes the robot start a traversal along the simulation, the players must follow through, and in the end, it will be said player's opponent's turn. The opponent would have to make the robot traverse a gadget not in the simulation. Players would be disincentivized to start a traversal along a closed path, because they will be the one stuck with no legal moves. So the simulation would act exactly like a locking 2-toggle in the

above reduction, giving us a straightforward reduction 2-player impartial motion planning with locking 2-toggles to 2-player impartial motion planning with any interacting $k$-tunnel reversible deterministic gadget.

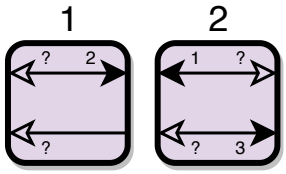



**Figure 17** Two tunnels that an interacting $k$-tunnel reversible deterministic gadget must have. Solid arrows indicate open traversals, hollow arrows with '?' indicate optionally open traversals, and absent arrows indicate closed traversals. State 3 could be any state, including 1 and 2.

First we simulate a 1-tunnel reversible deterministic gadget with a directed tunnel, as shown in Figure 18. The robot cannot cross from right to left. If it crosses from left to right, it may cross back (after traversing some other gadget, of course), and the path from left to right may optionally still be open, this time leading to whatever state. Note that it takes two traversals to cross the simulation, and that a closed path in state 1 of the gadget used in the simulation blocks the robot after 0 traversals.

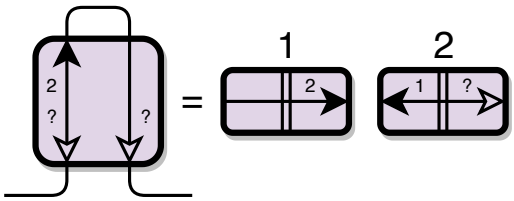



**Figure 18** Simulation of a 1-tunnel reversible deterministic gadget with a directed tunnel. We draw double bars crossing the 1-tunnel gadget as a reminder that it takes two traversals to cross.

Now we simulate the locking 2-toggle, as shown in Figure 19. The simulation currently simulates the locking 2-toggle in the nonleaf state. The robot can traverse from top right to top left or from bottom left to bottom right. The robot will get blocked after two traversals in an attempt to traverse from top left to top right or from bottom right to bottom left. If the robot traverses from top right to top left, the robot will be able to traverse from top left to top right (after traversing a different gadget). But an attempt to traverse from bottom left to bottom right gets the robot blocked after 0 traversals, thanks to the tunnel interaction in the left gadget, and an attempt to traverse from bottom right to bottom left or from top right to top left gets blocked after two traversals. So this would simulate a leaf state of the locking 2-toggle. The center gadget never becomes relevant for blocking, so we can argue by symmetry that traversing from bottom left to bottom right results in the other leaf state. Note that each path takes nine traversals to cross, so we have successfully simulated the locking 2-toggle meeting the constraints. This completes the proof. ◀

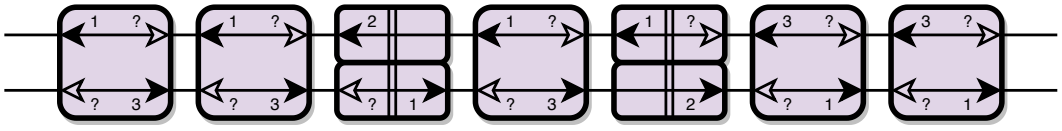



**Figure 19** Simulation of the locking 2-toggle, under the constraints.

By Lemma 5.1 and Theorem 5.3, it is EXPTIME-complete to determine whether Player 1 has a deterministic winning strategy in the 2-player impartial motion planning game with any interacting $k$-tunnel reversible deterministic gadget.

## 6 Open Problems

For 0-player motion planning, we leave as an open problem whether the finite-time reachability problem is undecidable for a smaller set of gadgets. In particular, we used gadgets that can separate one robot from the rest when they are all stuck at the same spot. Is the problem undecidable for gadgets without this ability? What about classes of gadgets that have already been studied such as self-closing doors or reversible, deterministic gadgets?

In the 0-player model with spawners we investigated a synchronous model for the robots where they all took turns making their moves. One could imagine asking about various asynchronous models of robot motion through the gadgets.

For 1-player multi-agent motion planning, we investigated robot reachability and multi-agent targeted reconfiguration. The hardness for both these problems relies on simulating Petri nets with a symmetric self-closing door. Do there exist reversible gadgets for which the problem is the same complexity? How does this relate to reversible Petri nets?

We also did not investigate spawners in the 2-player setting. It seems likely that this problems is Undecideable for many gadget; however, the 0-player and 1-player constructions do not obviously adapt to give this result.

Finally, in the 2-player impartial case, does the complexity change for other gadgets? Are there any gadgets for which finding a winning strategy is provably easier? What about cases where the impartial game is harder than the regular 2-player game?

## References


1 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Dyl an H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Korten, Ja yson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In Kevin Buchin and Éric Colin de Verdière, editors, *Proceedings of the 37th International Symposium on Computational Geometry*, LIPIcs, pages 10:1–10:20, 2021.

2 Robert M. Alaniz, Bin Fu, Timothy Gomez, Elise Grizzell, Andrew Rodriguez, Robert Schweller, and Tim Wylie. Reachability in restricted chemical reaction networks. *arXiv preprint*, 2022. `arXiv:2211.12603`.

3 Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020)*, Favignana, Italy, September 2020.

4 Joshua Ani, Lily Chung, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude. In *Proceedings of the 11th International Conference on Fun with Algorithms*, pages 2:1–2:30, Island of Favignana, Sicily, Italy, May–June 2022.

5 Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, Dylan H. Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. In Petra Mutzel, Md. Saidur Rahman, and Slamin, editors, *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation*, volume 13174 of *Lecture Notes in Computer Science*, pages 47–58, Jember, Indonesia, March 2022. `doi:10.1007/978-3-030-96731-4_5`.

6 Joshua Ani, Erik D. Demaine, Dylan Hendrickson, and Jayson Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. In Petra Mutzel, Md. Saidur Rahman, and Slamin, editors, *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation*, volume 13174 of *Lecture Notes in Computer Science*, pages 187–198, Jember, Indonesia, March 2022.

7 Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A Cantu, Robert Schweller, Luis Angel Garcia, and Tim Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2689–2708. SIAM, 2019.

8 David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie. Relocating units in robot swarms with uniform control signals is PSPACE-complete. *CCCG 2020*, 2020.

9 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1229–1240, 2022.

10 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*, pages 18:1–18:21, La Maddalena, Italy, June 2018.

11 Erik D. Demaine, Robert A. Hearn, Dylan Hendrickson, and Jayson Lynch. PSPACE-completeness of reversible deterministic systems. In *Proceedings of the 9th Conference on Machines, Computations and Universality*, pages 91–108, Debrecen, Hungary, August–September 2022.

12 Erik D. Demaine, Dylan Hendrickson, and Jayson Lynch. Toward a general theory of motion planning complexity: Characterizing which gadgets make games hard. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science*, Seattle, Washington, January 2020.

13 Javier Esparza. Decidability and complexity of petri net problems – An introduction. *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*, pages 374–428, 2005.

14 Dylan Hendrickson. Gadgets and gizmos: A formal model of simulation in the gadget framework for motion planning. Master's thesis, Massachusetts Institute of Technology, 2021.

15 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1241–1252, 2022.

16 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13. IEEE, 2019.

17 Jayson Lynch. *A framework for proving the computational intractability of motion planning problems*. PhD thesis, Massachusetts Institute of Technology, 2020.

18 Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc, 1967.

19 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

20 Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *Siam Journal on Computing*, 8(2):151–174, May 1979.

# Adaptive Collective Responses to Local Stimuli in Anonymous Dynamic Networks

**Shunhao Oh**
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

**Dana Randall**
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

**Andréa W. Richa**
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

## Abstract

We develop a framework for self-induced phase changes in programmable matter in which a collection of agents with limited computational and communication capabilities can collectively perform appropriate global tasks in response to local stimuli that dynamically appear and disappear. Agents reside on graph vertices, where each stimulus is only recognized locally, and agents communicate via token passing along edges to alert other agents to transition to an AWARE state when stimuli are present and an UNAWARE state when the stimuli disappear. We present an Adaptive Stimuli Algorithm that is robust to competing waves of messages as multiple stimuli change, possibly adversarially. Moreover, in addition to handling arbitrary stimulus dynamics, the algorithm can handle agents reconfiguring the connections (edges) of the graph over time in a controlled way.

As an application, we show how this Adaptive Stimuli Algorithm on reconfigurable graphs can be used to solve the *foraging problem*, where food sources may be discovered, removed, or shifted at arbitrary times. We would like the agents to consistently self-organize, using only local interactions, such that if the food remains in a position long enough, the agents transition to a *gather phase* in which many collectively form a single large component with small perimeter around the food. Alternatively, if no food source has existed recently, the agents should undergo a self-induced phase change and switch to a *search phase* in which they distribute themselves randomly throughout the lattice region to search for food. Unlike previous approaches to foraging, this process is indefinitely repeatable, withstanding competing waves of messages that may interfere with each other. Like a physical phase change, microscopic changes such as the deletion or addition of a single food source trigger these macroscopic, system-wide transitions as agents share information about the environment and respond locally to get the desired collective response.

**2012 ACM Subject Classification** Theory of computation → Self-organization; Theory of computation → Random walks and Markov chains

**Keywords and phrases** Dynamic networks, adaptive stimuli, foraging, self-organizing particle systems, programmable matter

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.6

**Related Version** *Full Version*: https://arxiv.org/abs/2304.12771

**Funding** *Shunhao Oh*: Funded in part by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.
*Dana Randall*: Supported by the National Science Foundation (NSF) award CCF-2106687 and by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.
*Andréa W. Richa*: Supported in part by the National Science Foundation (NSF) award CCF-2106917 and by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.

**Acknowledgements** The authors thank the anonymous reviewers for useful feedback on the presentation.

© Shunhao Oh, Dana Randall, and Andréa W. Richa;
licensed under Creative Commons License CC-BY 4.0



## 1 Introduction

Self-organizing collective behavior of interacting agents is a fundamental, nearly ubiquitous phenomenon across fields, reliably producing rich and complex coordination. In nature, examples at the micro- and nano-scales include coordinating cells, including our own immune system or self-repairing tissue (e.g., [1]), and bacterial colonies (e.g., [26, 31]); at the macro-scale it can represent flocks of birds [7], shoals of fish aggregating to intimidate predators [27], fire ants forming rafts to survive floods [29], and human societal dynamics such as segregation [34]. Common characteristic of these disparate systems is that they are all self-actuated and respond to simple, local environmental stimuli to collectively change the ensemble's behavior.

In 1991, Toffoli and Margolus coined *programmable matter* to realize a physical computing medium composed of simple, homogeneous entities that can dynamically alter its physical properties in a programmable fashion, controlled either by user input or its own autonomous sensing of its environment [36]. There are formidable challenges to realizing such collective tasks and many researchers in distributed computing and swarm and modular robotics have investigated how such small, simply programmed entities can coordinate to solve complex tasks and exhibit useful emergent collective behavior (e.g., [32]). A more ambitious goal, suggested by self-organizing collective systems in nature, is to design programmable matter systems of *self-actuated* individuals that *autonomously respond to continuous, local changes in their environment.*

**The Dynamic Stimuli Problem.** As a distributed framework for agents collectively self-organizing in response to changing stimuli, we consider the *dynamic stimuli problem* in which we have a large number of agents that collectively respond to local signals or stimuli. These agents have limited computational capabilities and each only communicates with a small set of immediate neighbors. We represent these agents via a dynamic graph $G$ on $n$ vertices, where the agents reside at the vertices and edges represent pairs that can perceive and interact with each other. At arbitrary points of time, that may be adversarially chosen, *stimuli* dynamically appear and disappear at the vertices of $G$ – these can be a threat, such as an unexpected predator, or an opportunity, such as new food or energy resources.

An agent present at the same vertex as a stimulus acts as a *witness* and alerts other agents. If any agent continues to witness some stimulus over an extended period of time, we want all agents to eventually be alerted, switching to the AWARE state; on the other hand, once witnesses stop sensing a stimulus for long enough, all agents should return to the UNAWARE state. Such collective state changes may repeat indefinitely as stimuli appear and disappear over time. Converging to these two global states enables agents to carry out *differing behaviors in the presence or absence of stimuli*, as observed by the respective witnesses. As a notable and challenging example, in *the foraging problem,* "food" may appear or disappear at arbitrary locations in the graph over time and we would like the collective to *gather around food* (also known as *dynamic free aggregation*) or *disperse in search of new sources*, depending on the whether or not an active food source has been identified. Cannon *et al.* [5] showed how computationally limited agents can be made to gather or disperse; however there the desired goal, aggregation or dispersion, is fixed in advance and the algorithm cannot easily be adapted to move between these according to changing needs.

In addition to the stimuli dynamics, we assume that the agents may *reconfigure the connections (edges) of the graph $G$ over time*, but in a controlled way that still allows the agents to successfully manage the waves of state changes. In a nutshell, $G$ is *reconfigurable*

over time if it maintains recurring local connectivity of the AWARE agents (i.e., if it makes sure that the 1-hop neighborhood sets of AWARE agents stay connected over time), as its edge set changes. The edge dynamics may be fully in the control of an adversary, or may be controlled by the agents themselves, depending on the context (e.g., in the foraging problem presented in Section 5, the agents control the edge dynamics).

In this framework, we assume that at all times there are at most a constant number $w$ of stimuli present at the vertices of $G$. Agents are anonymous and each acts as a finite automaton with constant-size memory, constant degree, and no access to global information other than $w$ and a constant upper bound $\Delta$ on the maximum degree. Individual agents are activated according to their own Poisson clocks and perform instantaneous actions upon activation, a standard way to allow sites to update independently and asynchronously (since the probability two Poisson clocks tick at the exact same instant in time is negligible). For ease of discussion, we may assume the Poisson clocks have the same rate, so this model is equivalent to a *random sequential scheduler* that chooses an agent uniformly at random to be activated at discrete iterations $t \in \{1, 2, 3, \ldots\}$. We denote by $G_t$ the configuration of the reconfigurable graph at iteration $t$. When *activated* at iteration $t$, an agent perceives its own state and the states of its current neighbors in $G_t$, performs a bounded amount of computation, and can change its own and its neighbors states, including any "tokens" (i.e., constant size messages received or sent).[1] At each iteration $t \in \{1, 2, 3, \ldots\}$, we denote by $\mathcal{W}_t \subseteq V$ the set of witnesses. The sets $\mathcal{W}_t$ can change arbitrarily (adversarially) over time, but $|\mathcal{W}_t|$ is always bounded by the constant $w$, for all $t$, since $w$ is an upper on the number of concurrent stimuli.

**Overview of Results.** Our contributions are two-fold. First, we present an efficient, robust algorithm for the dynamic stimuli problem for a class of reconfigurable graphs. (The precise details of what constitutes a *valid reconfigurable graph* will be given in Section 4.) Whenever an agent encounters a new stimulus, the entire collective efficiently transforms to the AWARE state, so the agents can implement an appropriate collective response. After a stimulus vanishes, they all return to the UNAWARE state, recovering their neutral collective behavior.

We show that the system will always converge to the appropriate state (i.e., AWARE or UNAWARE) once the stimuli stabilize for a sufficient period of time. Specifically, if there are no witnesses in the system for a sufficient period of time, then all agents in the Adaptive Stimuli Algorithm will reach and remain in the UNAWARE state in $O(n^2)$ expected time (Theorem 14). Likewise, if the set of witnesses (and stimuli) remains unchanged for a sufficient period of time, all agents will reach and remain in the AWARE state in expected time that is a polynomial in $n$ and the "recurring rate" of $G$, which captures how frequently disconnected vertices come back in contact with each other (Theorem 15). In particular, if $G$ is static or if $G_t$ is connected, for all $t$, the expected convergence time until all agents transition to the AWARE state is $O(n^5)$ (Theorem 4) and $O(n^6 \log n)$ (Corollary 16), respectively. Moreover, the system can recover if the witness set changes before the system converges to the AWARE or UNAWARE states.

While the arguments are simpler for sequences of connected graphs generated by, say, an oblivious adversary, the extension to the broader class of reconfigurable graphs includes graphs possibly given by non-oblivious adversaries that may occasionally disconnect. This

[1] Since we assume a sequential scheduler, such an action is justified; in the presence of a stronger adversarial scheduler, e.g., the asynchronous scheduler, one would need a more detailed message passing mechanism to ensure the transfer of tokens between agents, and resulting changes in their states.

generalization provides more flexibility for agents in the AWARE and UNAWARE states to implement more complex behaviors, as was used for applying the Adaptive Stimuli Algorithm to foraging, which we describe next.

Our second main contribution is the first efficient algorithm for the *foraging problem*, where food dynamically appears and disappears over time at arbitrary sites on a finite $\sqrt{N} \times \sqrt{N}$ region of the triangular lattice. Agents want to gather around any discovered food source (also known as *dynamic free aggregation*) or disperse in search of food. The algorithm of Cannon *et al.* [5, 25] uses insights from the high and low temperature phases of the ferromagnetic Ising model from statistical physics to provably achieve either desired collective response: there is a preset global parameter $\lambda$ related to inverse temperature, and the algorithm provably achieves aggregation when $\lambda$ is sufficiently high and dispersion when sufficiently low. We show here that by applying the Adaptive Stimuli Algorithm, the phase change (or bifurcation) for aggregation and dispersion can be self-modulated based on local environmental cues that are communicated through the collective to induce desirable system-wide behaviors in polynomial time in $n$ and $N$, as stated in Theorems 23 and 24.

Collectively transitioning between AWARE and UNAWARE states enables agents to *correctly self-regulate system-wide adjustments* in their bias parameters when one or more agents notice the presence or depletion of food to induce the appropriate global coordination to provably transition the collective between macro-modes when required. We believe other collective behaviors exhibiting emergent bifurcations, including separation/integration [4] and alignment/nonalignment [22], can be similarly self-modulated.

The distinction between polynomial and super-polynomial running times is significant here because our algorithms necessarily make use of competing broadcast waves to propagate commands to change states. A naive implementation of such a broadcast system may put us in situations where neither type of wave gets to complete its propagation cycle. This may continue for an unknown amount of time, so the agents may fail to reach an agreement on their state. The carefully engineered token passing mechanism ensures that when a stimulus has been removed, the *rate at which the agents "reset" to the* UNAWARE *state outpaces the rate at which the cluster of* AWARE *agents may continue to grow*, ensuring that the newer broadcast wave always supersedes previous ones and completes in expected polynomial time. Moreover, while a stimulus is present, there is a continuous *probabilistic generation of tokens that move according to a $d_{max}$-random walk* among AWARE agents until they find a new agent to become AWARE, thus ensuring the successful convergence to the AWARE state in expected polynomial time.

**Related work.** Dynamic networks have been of growing interest recently and have spawned several model variants (see, e.g., the surveys in [6] and [24]). There is also a vast literature on broadcasting, or information dissemination, in both static and dynamic networks (e.g., [23, 20, 8, 15, 14]), where one would like to disseminate $k$ messages to all nodes of a graph $G$, usually with unique token ids and $k \leq n$, polylog memory at the nodes (which may also have unique ids), and often nodes' knowledge of $k$ and possibly also of $n$. Note that any of these assumptions violates our agents' memory or computational capabilities. Moreover, since our collective state-changing process runs indefinitely, any naive adaptations of these algorithms would need that $k \to \infty$ to ensure that with any sequence of broadcast waves, the latest always wins.

Broadcast algorithms that do not explicitly keep any information on $k$ (number of tokens or broadcast waves) or $n$ would be more amenable to our agents. Amnesiac flooding is one such broadcast algorithm that works on a network of anonymous nodes without keeping any

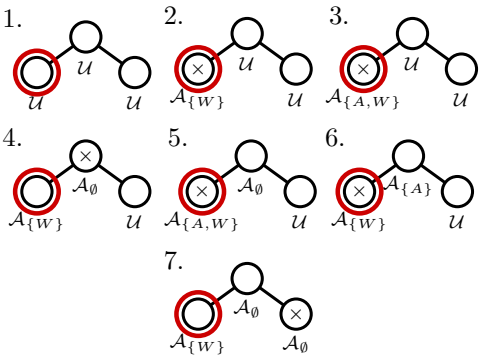



**Figure 1** Illustration of how the presence of a witness (circled in red) gradually converts all agents to the AWARE state through the distribution of alert tokens. The agent with the "×" is the agent activated in that step.

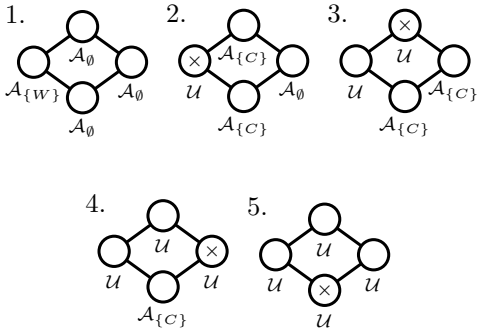



**Figure 2** Illustration of how all-clear tokens are broadcast from an agent with the witness flag set that is no longer a witness (the leftmost agent). The agent with the "×" is the agent activated in that step.

state or other information as the broadcast progresses. In [21], Hussak and Trehan show that amnesiac flooding will always terminate in a static network under synchronous message passing, but may fail on a dynamic network or with non-synchronous executions.

Many studies in self-actuated systems take inspiration from emergent behavior in social insects, but either lack rigorous mathematical foundations explaining the generality and limitations as sizes scale (see, e.g., [19, 18, 9, 38]), often approaching the thermodynamic limits of computing [37] and power [10], or rely on long-range signaling, such as microphones or line-of-sight sensors [35, 16, 17, 30]. Some recent work on stochastic approaches modeled after systems from particle physics has been made rigorous, but only when a single, static goal is desired [5, 4, 25, 2, 33, 22].

## 2 The Adaptive Stimuli Algorithm

The *Adaptive Stimuli Algorithm* is designed to efficiently respond to dynamic local stimuli that indefinitely appear and disappear at the vertices of $G$. Recall the goal of this algorithm is to allow the collective to converge to the AWARE state whenever a stimulus is witnessed for long enough and to the UNAWARE state if no stimulus has been detected recently. The algorithm converges in expected polynomial time in both scenarios under a reconfigurable dynamic setting, as we show in Sections 3-4, even as the process repeats indefinitely.

All agents know two parameters of the system: $\Delta \geq 1$, an upper bound on the maximum degree of the graph, and $w \geq 1$, an upper bound on the size of the witness sets $\mathcal{W}_t$ at all times $t$ (which is needed to determine the probability $p < 1/w$ for some agents to change states or generate certain tokens). Our algorithm defines a carefully balanced token passing mechanism, where a *token* is a constant-size piece of information: Upon activation, an AWARE witness $u$ continuously generates *alert tokens* one at a time, with probability $p$, which will each move through a *random walk over* AWARE *agents until they come in contact with a neighboring* UNAWARE *agent* $u$: The token is then consumed and $u$ changes its state to AWARE (Figure 1). On the other hand, if a witness notices that its co-located stimulus has disappeared, it will initiate an all-clear token broadcast wave which will proceed through agents in the AWARE state, switching those to UNAWARE (Figure 2).

**Algorithm 1** Adaptive Stimuli Algorithm.

**procedure** The Adaptive-Stimuli-Algorithm($u$)
  Let $p < 1/w \in (0,1)$
  $u.isWitness \leftarrow$True if $u$ is a witness, else $u.isWitness \leftarrow$False
  **if** $u.isWitness$ and $u.state \notin \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$ **then**
    With probability $p$, $u.state \leftarrow \mathcal{A}_{\{W\}}$ ▷ $u$ becomes Aware witness with prob. $p$
  **else if** $\neg u.isWitness$ and $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$ **then** ▷ stimulus no longer at $u$
    **for** each $v \in N_{\mathcal{A}}(u)$ **do** ▷ $N_{\mathcal{A}}(u)$ = current Aware neighbors of $u$
      $v.state \leftarrow \mathcal{A}_{\{C\}}$ ▷ all-clear token broadcast to Aware neighbors of $u$
    $u.state \leftarrow \mathcal{U}$
  **else**
    **switch** $u.state$ **do**
      **case** $\mathcal{U}$:
        **if** $\exists v \in N_{\mathcal{A}}(u), v.state = \mathcal{A}_S \in \{\mathcal{A}_{\{A\}}, \mathcal{A}_{\{A,W\}}\}$ **then** ▷ $v$ has alert token
          $v.state \leftarrow \mathcal{A}_{S\setminus\{A\}}$ ▷ $v$ consumes alert token
          $u.state \leftarrow \mathcal{A}_{\emptyset}$ ▷ $u$ becomes aware
      **case** $\mathcal{A}_{\{A\}}$ or $\mathcal{A}_{\{A,W\}}$:
        $x \leftarrow$ random value in $[0,1]$
        **if** $x \leq d_G(u)/\Delta$ **then** ▷ $d_{max}$-random walk
          $v \leftarrow$ random neighbor of $u$
          **if** $v.state = \mathcal{A}_{S'} \in \{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{W\}}\}$ **then** ▷ Aware state, no alert token
            Let $u.state = \mathcal{A}_S$; $u.state \leftarrow \mathcal{A}_{S\setminus\{A\}}$ ▷ $u$ sends alert token to $v$
            $v.state \leftarrow \mathcal{A}_{S'\cup\{A\}}$ ▷ $v$ receives alert token
      **case** $\mathcal{A}_{\{W\}}$:
        With probability $p$, $u.state \leftarrow \mathcal{A}_{\{A,W\}}$ ▷ generate alert token with prob. $p$
      **case** $\mathcal{A}_{\{C\}}$:
        **for** each $v \in N_{\mathcal{A}}(u)$ **do**
          $v.state \leftarrow \mathcal{A}_{\{C\}}$ ▷ $u$ broadcasts all-clear token to all aware neighbors
        $u.state \leftarrow \mathcal{U}$

The differences between these two carefully crafted token passing mechanisms allow us to ensure that whenever there has been no stimulus in the network for long enough, the rate at which the agents deterministically learn this (through broadcasts of all-clear tokens) and become Unaware always outpaces the probabilistic rate at which the cluster of Aware agents may still continue to grow. Thus, the Unaware broadcast wave will always outpace any residual Aware waves in the system and will allow the collective to correctly converge to the desired Unaware state. On the other hand, if the witness set is non-empty and remains stable for a long enough period of time, all agents will eventually switch to the Aware state since, after some time, there will be no all-clear tokens in $G$, as Unaware agents do not ever generate or broadcast tokens.

To define the Adaptive Stimuli Algorithm, we utilize the following flags and states:

- **Alert token flag** ($A$): used to indicate that an agent has an alert token.
- **All-clear token flag** ($C$): used to indicate that the agent has an all-clear token.
- **Witness flag** ($W$): used to indicate that the agent is a witness.
- **States**: The Unaware state is denoted by $\mathcal{U}$, while $\{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{A\}}, \mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}, \mathcal{A}_{\{C\}}\}$ denote the Aware states. In the Aware states, the subscript denotes the subset of flags that are currently set. Note that the all-clear token flag is only set when the other two are not, giving us six distinct states in total.

Algorithm 1 formalizes the actions executed by each agent $u$ when activated. Agent $u$'s actions depend on the current state it is in and whether it senses a stimulus. We describe the behavior for each of the possible cases below:

- **Non-matching witness flag**: This is a special case that occurs if $u$ is currently a witness to some stimuli but its witness flag has not been set yet, or if $u$ has its witness flag set but is no longer a witness. This case takes priority over all the other possible cases, since $u$ cannot take any action before its witness status and its state match. If $u$ is a witness but does not have the witness flag set, then with probability $p$, switch $u$ to state $\mathcal{A}_{\{W\}}$. On the other hand, if $u$ is not a witness but has the witness flag set, switch $u$ to UNAWARE (by setting $u.state = \mathcal{U}$) and broadcast an all-clear token to all of $u$'s AWARE neighbors (this token overrides any other tokens the neighbors may have).
- **Unaware state ($\mathcal{U}$)**: If $u$ has an AWARE neighbor $v$ with an alert token (i.e., $v.state \in \{\mathcal{A}_{\{A\}}, \mathcal{A}_{\{A,W\}}\}$), then $u$ consumes the alert token from $v$ (by setting $v$'s alert token flag to FALSE) and switches to the state $\mathcal{A}_{\emptyset}$.
- **Aware state with alert token ($\mathcal{A}_{\{A\}}$, $\mathcal{A}_{\{A,W\}}$)**: Pick a random neighbor of $u$ such that each neighbor is picked with probability $\frac{1}{\Delta}$, with a probability $1 - \frac{d_G(u)}{\Delta}$ of staying at $u$ (this executes a $d_{max}$-random walk [13]). If an AWARE neighbor $v$ is picked and $v$ does not have an alert nor an all-clear token (that is, $v.state \in \{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{W\}}\}$), move the alert token to $v$ by toggling the alert token flags on both $u$ and $v$.
- **Aware state with witness flag but without an alert token ($\mathcal{A}_{\{W\}}$)**: With probability $p$, $u$ generates a new alert token by switching to state $\mathcal{A}_{\{A,W\}}$.
- **Aware state with all-clear token ($\mathcal{A}_{\{C\}}$)**: Switch $u$ to the UNAWARE state $\mathcal{U}$ and broadcast the all-clear token to all of its AWARE state neighbors.

The use of a $d_{max}$-random walk instead of regular random walk (Line 18 of Algorithm 1) normalizes the probabilities of transitioning along an edge by the maximum degree of the nodes (or a constant upper bound on that), so that these transition probabilities cannot change during the evolution of the graph. A $d_{max}$-random walk has polynomial hitting time on any connected dynamic network (while a regular random walk might not) [3].

## 3 Static graph topologies

For simplicity, we will first state and prove our results for *static connected graph topologies* (Theorems 3 and 4), where the edge set never changes and the dynamics are only due to the placement of stimuli. In Section 4, we show that the same proofs apply with little modification to the reconfigurable case as well.

In order to define our main theorems, we must first define the *state invariant*, that we know holds from an initial configuration where every agent is initialized in the UNAWARE state, as we show Lemma 2. In the remainder of this paper, a *component* will refer to a connected component of the subgraph induced in $G$ by the set of AWARE agents.

▶ **Definition 1** (State Invariant). *We say a component satisfies the* state invariant *if it contains at least one agent in the states $\mathcal{A}_{\{W\}}$, $\mathcal{A}_{\{A,W\}}$ or $\mathcal{A}_{\{C\}}$. A configuration satisfies the state invariant if every component (if any) of the configuration satisfies the state invariant.*

▶ **Lemma 2.** *If the current configuration satisfies the state invariant, then all subsequent configurations reachable by Algorithm 1 also satisfy the state invariant.*

**Proof.** Starting from configuration where the state invariant currently holds, let $u$ be the next agent to be activated. If $u$ is a witness but $u.state \notin \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$, switching $u$ to state $\mathcal{A}_{\{W\}}$ does not affect the state invariant. Conversely, if $u$ is not a witness, but

$u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$ switching $u$ to state $\mathcal{U}$ can potentially split the component it is in into multiple components. However, as all neighbors of $u$ will also be set to state $\mathcal{A}_{\{C\}}$, each of these new components will contain an agent in state $\mathcal{A}_{\{C\}}$.

Otherwise, if $u.state = \mathcal{U}$, it only switches to the AWARE state if it neighbors another AWARE agent. As the component $u$ joins must contain an agent in states $\mathcal{A}_{\{W\}}$, $\mathcal{A}_{\{A,W\}}$ or $\mathcal{A}_{\{C\}}$, the new configuration will continue to satisfy the state invariant. If $u.state = \mathcal{A}_{\{C\}}$, similar to the earlier case where $u$ is not a witness but $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}\}$, activating $u$ may split the component it is in. As before, all neighbors of $u$ will be set to state $\mathcal{A}_{\{C\}}$, so each of these newly created components satisfy the state invariant. The remaining possible cases only toggle the alert token flag, which does not affect the state invariant. ◀

This allows us to state our main results in the static graph setting. We let $T \in \mathbb{N}$ represent the time when the stimuli stabilize long enough to converge (where $T$ is unknown to the agents) and show that we will have efficient convergence. Without loss of generality, for the sake of our proofs, we will assume that $\mathcal{W}_t = \mathcal{W}_T$, for all $t \geq T$, although this is really just representing a phase where the stimuli are stable.

▶ **Theorem 3.** *Starting from any configuration satisfying the state invariant over a static connected graph topology $G$, if $|\mathcal{W}_T| = 0$, then all agents will reach and remain in the* UNAWARE *state in $O(n^2)$ expected iterations, after time $T$.*

▶ **Theorem 4.** *Starting from any configuration satisfying the state invariant over a static connected graph topology $G$, if $|\mathcal{W}_T| > 0$, then all agents will reach and stay in the* AWARE *state in $O(n^5)$ expected iterations, after time $T$.*

The proof of these theorems relies on carefully eliminating residual aware agents from previous broadcasts. These agents have yet to receive an all-clear token and thus will take some time before they can return to the UNAWARE state.

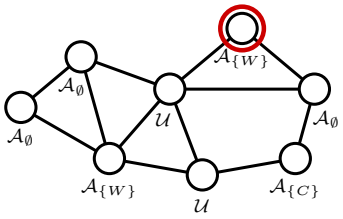



**Figure 3** A configuration with two residual components. The component on the left is a residual component despite having a witness in it because it contains an agent with the all-clear flag; the component on the right is a residual component since it contains an agent in state $\mathcal{A}_{\{C\}}$.

▶ **Definition 5** (Residuals). *A* residual component *is a component that satisfies at least one of the following two criteria:*

1. *It contains an agent in state $\mathcal{A}_{\{C\}}$.*
2. *It contains an agent in state $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ that is not a witness.*

*We call agents belonging to residual components* residuals.

When starting from arbitrary configurations, there is likely to be a large number of residual components that need to be cleared out. Residuals are problematic as they do no stay in the aware state in the long term. This means they do not actually contribute to the main cluster of AWARE state agents, and the presence of residuals causes even more residuals to form. Furthermore, later on when we allow the graph to reconfigure itself, residuals may

obstruct UNAWARE state agents from coming into contact with non-residual components, which may prevent alert tokens from reaching them. Our main tool to show that all residuals eventually vanish is a *potential function* that decreases more quickly than it increases.

▶ **Definition 6** (Potential). *For a configuration $\sigma$, we define its potential $\Phi(\sigma)$ as*

$$\Phi(\sigma) := \Phi_A(\sigma) + \Phi_{AT}(\sigma)$$

*where $\Phi_A(\sigma)$ and $\Phi_{AT}(\sigma)$ represent total the number of AWARE agents and the number of AWARE agents with an alert token respectively.*

We use the following lemma to guarantee, in Lemma 8, that the expected number of steps before all residuals are removed is polynomial.

▶ **Lemma 7.** *Assume $n \geq 2$, and let $0 < \eta < 1$. Consider two random sequences of probabilities $(p_t)_{t\in\mathbb{N}_{\geq 0}}$ and $(q_t)_{t\in\mathbb{N}_{\geq 0}}$, with the properties that $\frac{1}{n} \leq p_t \leq 1$ and $0 \leq q_t \leq \frac{\eta}{n}$, and $p_t + q_t \leq 1$. Now consider a sequence $(X_t)_{t\in\mathbb{N}_{\geq 0}}$, where*

$$X_{t+1} \begin{cases} \leq X_t - 1 & \text{with probability } p_t \\ = X_t + 1 & \text{with probability } q_t \\ = X_t & \text{with probability } 1 - p_t - q_t. \end{cases}.$$

*Then $\mathbb{E}\left[\min\{t \geq 0 \mid X_t = 0\}\right] \leq \frac{nX_0}{1-\eta}$.*

**Proof.** For each $k \in \mathbb{N}_{\geq 0}$, we define a random sequence $(Y_t^{(k)})_{t\in\mathbb{N}_{\geq 0}}$ such that $Y_0^{(k)} = k$ and

$$Y_{t+1}^{(k)} \begin{cases} = Y_t^{(k)} - 1 & \text{with probability } \frac{1}{n} \\ = Y_t^{(k)} + 1 & \text{with probability } \frac{\eta}{n} \\ = Y_t^{(k)} & \text{otherwise.} \end{cases}$$

For each such $k$, let $S_k := \mathbb{E}\left[\min\{t \geq 0 \mid Y_t^{(k)} = 0\}\right]$. Let $T_{k+1}^{(k)} = 0$ and for $i \in \{k, k-1, \ldots, 1\}$, let $T_i^{(k)} = \min\{t \geq 0 \mid X_t^{(k)} \leq i\} - T_{i+1}^{(k)}$ denote the number of time steps after $T_{i+1}^{(k)}$ before the first time step $t$ where $Y_t^{(k)} \leq i$. We observe that each $T_i^{(k)}$ is identically distributed, with $\mathbb{E}T_i^{(k)} = \mathbb{E}T_1^{(k)} = S_1$. Also, we observe that $S_k = \mathbb{E}[\sum_{i=1}^{k} T_i^{(k)}]$, so $S_k = k \cdot S_1$ for all $k$. We can thus compute $S_1$ by conditioning on the first step:

$$S_1 = \frac{1}{n}(1) + \frac{\eta}{n}(S_2 + 1) + \left(1 - \frac{1+\eta}{n}\right)(S_1 + 1) = 1 + \frac{\eta}{n} \cdot 2S_1 + \left(1 - \frac{1+\eta}{n}\right) S_1$$

This implies $S_1 = \frac{n}{1-\eta}$ and thus $\mathbb{E}\left[\min\{t \geq 0 \mid Y_t^{(k)} = 0\}\right] = S_k = \frac{kn}{1-\eta}$.

We can then couple $(X_t)_{t\in\mathbb{N}_{\geq 0}}$ and $(Y_t^{(X_0)})_{t\in\mathbb{N}_{\geq 0}}$ in a way such that $Y_{t+1}^{(X_0)} = Y_t^{(X_0)} + 1$ whenever $X_{t+1} > X_t$, and $Y_{t+1}^{(X_0)} = Y_t^{(X_0)} - 1$ whenever $X_{t+1} < X_t$. We thus have $Y_t^{(X_0)} \geq X_t$ always, and so $\mathbb{E}\left[\min\{t \geq 0 \mid X_t = 0\}\right] \leq \mathbb{E}\left[\min\{t \geq 0 \mid Y_t^{(X_0)} = 0\}\right] \leq \frac{kX_0}{(1-\eta)}$. ◀

We show in Lemma 8 that after a polynomial number of steps in expectation, we will reach a configuration with no residual components.

▶ **Lemma 8.** *We start from a configuration satisfying the state invariant over a static connected graph $G$ with no more than $w$ witnesses at any point. Then the expected number of steps before we reach a configuration with no residual components is at most $2n^2/(1-wp)$.*

**Proof.** We apply Lemma 7 to the sequence of potentials $(\Phi(\sigma_t))_{t\in\mathbb{N}_{\geq 0}}$ where $\sigma_t$ is the configuration after iteration $t$. As long as there exists a residual component, there will be at least one agent will switch to the UNAWARE state on activation. This gives a probability of at least $1/n$ in any iteration of decreasing the current potential by at least 1.

There are only two ways for the potential to increase. The first is when a new alert token is generated by a witness, and the second is when an UNAWARE witness switches to an AWARE state. The activation of a witness thus increases the current potential by exactly 1, with probability $p$. As there are at most $w$ witnesses, this happens with probability at most $wp/n < 1/n$. Note that the consumption of an alert token to add a new AWARE state agent to a residual component does not change the current potential. Neither does switching agents to the all-clear token state affect the potential.

By Lemma 7, as $\Phi(\sigma_0) \leq 2n$, within $2n^2/(1-wp)$ steps in expectation, we will either reach a configuration $\sigma$ with $\Phi(\sigma) = 0$, or a configuration with no residual components, whichever comes first. Note that if $\Phi(\sigma) = 0$, then $\sigma$ cannot have any residual components, completing the proof. ◀

We now show that as long as no agent is removed from the witness set, after all residual components are eliminated, no new ones will be generated:

▶ **Lemma 9.** *We start from a configuration satisfying the state invariant over a static connected graph $G$, and assume that no agent will be removed from the witness set from the current point on. If there are currently no residuals, then a residual cannot be generated.*

**Proof.** With no residual components in the current iteration, there will be no agents in state $\mathcal{A}_{\{C\}}$, and all agents in state $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ will be witnesses. This means that no agent on activation will switch another agent to the $\mathcal{A}_{\{C\}}$ state. All agents in states $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ will continue to be witnesses by assumption of the lemma, and agents will only switch to states $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ if they are witnesses. Thus no component will be residual in the next iteration. ◀

When $|\mathcal{W}_T| = 0$ and no witnesses exist in the long term, if the state invariant holds, then all agents will be in the UNAWARE state, giving us Theorem 3. On the other hand, in order to show Theorem 4,if a witness persists in the long term , we need to show that all agents eventually switch to the AWARE state. Lemma 10 establishes this as without residuals, no AWARE state agent can revert to the UNAWARE state.

▶ **Lemma 10.** *We start from a configuration satisfying the state invariant over a static connected graph $G$, and assume that there are no residual agents, the witness set is nonempty and no agent will be removed from the witness set from the current point on. Then the expected number of iterations before the next agent switches from the UNAWARE to the AWARE state is at most $O(n^4)$.*

**Proof.** In the case where there is no AWARE agent, there must be an UNAWARE witness, which on activation switches to the AWARE state with probability $p$. The expected time before this happens is no more than $O(\frac{n}{p})$. Thus for the rest of the proof, we may assume every witness is already in the AWARE state with the witness flag set.

If we have a bound on the expected time before we reach a configuration where every AWARE agent holds an alert token, all it takes following that is for any UNAWARE agent to be activated while holding an alert token. As the graph $G$ is connected with least one UNAWARE and one AWARE agent, there will be some UNAWARE agents neighboring an AWARE agent, so the expected number of iterations before this occurs will is $O(n)$.

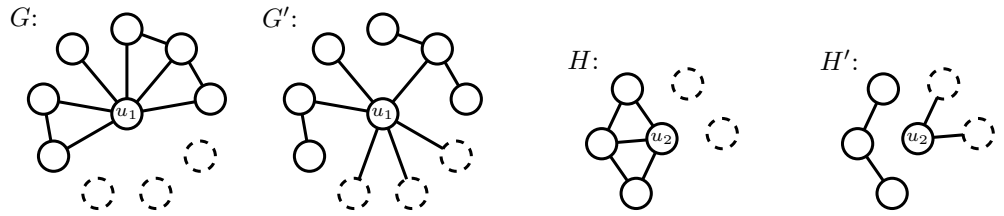



**Figure 4** Two locally connected reconfigurations of the vertices $u_1$ (from $G$ to $G'$) and $u_2$ (from $H$ to $H'$) respectively, where only the AWARE neighbors of the reconfigured vertex are shown. Vertices with dashed outlines are newly introduced neighbors.

We now bound the amount of time it takes for all AWARE state agents to hold alert tokens. As an agent can only hold one alert token at a time, a new alert token can only be generated when a witness does not hold an alert token. Assume that there is at least one AWARE state agent that does not have an alert token. Mark one such agent. Suppose that $u$ is the marked agent and that the next agent $v$ to be activated is a neighbor of $u$. If $v$ has an alert token and $v$ randomly chooses $u$ as its outgoing neighbor (per the algorithm), then $v$ transfers its alert token to $u$ and receives the mark from $u$. Otherwise, for the sake of our analysis, we still have $v$ pick an outgoing neighbor at random and receive the mark if the chosen neighbor is $u$.

The mark moving in this manner is equivalent to following a $d_{max}$-random walk over the subgraph induced by the AWARE agents, which is static as long as no new agents are switching to the AWARE state. As the configuration satisfies the state invariant and there are no residuals, the component of AWARE agents the mark is in must contain at least one witness. The worst case hitting time of the $d_{max}$-random walk over this subgraph is $O(n^2)$, and thus the expected number of iterations before the mark lands on a witness is $O(n^3)$, allowing a new alert token to be generated (this new alert token is generated with a constant probability $p \in (0, 1)$). As there are at most $n$ AWARE agents, all AWARE agents will be holding an alert token after $O(n^4)$ iterations in expectation, which translates to an expected time bound of $O(n^4)$ before a new AWARE agent is added. Note that this is a loose bound - the bound has not been optimized for clarity of explanation. ◀

## 4 Reconfigurable topologies

We show that the same results hold if we allow some degree of reconfigurability of the edge set and relax the requirement that the graph must be connected. This gives us enough flexibility to implement a wider range of behaviors, like free aggregation or compression and dispersion in Section 5. When needed, we may refer to this as the *reconfigurable dynamic stimuli problem*, in order to clearly differentiate from the dynamic stimuli problem on static graphs that we considered in Section 3.

Instead of a static graph $G$, as we considered in Section 3, we now allow the edge set of the graph to be locally modified over time. These reconfigurations can be initiated by the agents themselves or controlled by an adversary, and they can be randomized or deterministic, but we require some restrictions on what reconfigurations are allowed, and what information an algorithm carrying out these reconfigurations may have access to. This will result in a restricted class of dynamic graphs, but will be general enough to be applied to the problem of foraging that we describe in Section 5.

The basic primitive for (local) reconfiguration of our graph by an agent $u$ is replacing the edges incident to vertex $u$ with new edges. We call this a reconfiguration of vertex $u$ and define local connectivity to formalize which reconfigurations are allowed.

▶ **Definition 11** (Locally Connected Reconfigurations). *For any graph $G = (V, E)$, let $G'$ be a the graph resulting from a reconfiguration of vertex $u \in V$ and let $N_{\mathcal{A}}(u)$ be the* Aware *neighbors of $u$ in $G$. We say that this reconfiguration is locally connected if $u$ has at least one* Aware *neighbor in $G'$ and if for every pair of vertices $v_1, v_2$ in $N_{\mathcal{A}}(u)$ with a path from $v_1$ to $v_2$ in the induced subgraph $G[N_{\mathcal{A}}(u) \cup \{u\}]$, there is also a path from $v_1$ to $v_2$ in $G'[N_{\mathcal{A}}(u) \cup \{u\}]$.*

Examples of locally connected reconfigurations are given in Figure 4.

In the reconfigurable dynamic stimuli problem, we want to be able to define reconfiguration behaviors for agents in the Aware (without all-clear token) and Unaware states. To define what reconfigurations of an agent $u$ are valid, we group our set of agent states into three subsets which we refer to as *behavior groups*. The three behavior groups are $\widehat{\mathcal{U}} := \{\mathcal{U}\}$, $\widehat{\mathcal{M}} := \{\mathcal{A}_\emptyset, \mathcal{A}_{\{A\}}\}$ and $\widehat{\mathcal{I}} := \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}, \mathcal{A}_{\{C\}}\}$, which we call Unaware, Mobile and Immobile respectively (the latter two referring to Aware state agents which are and are not allowed to change their neighboring edges respectively). We say that a locally connected reconfiguration of an agent $u$ is *valid* if it is not allowed to reconfigure in the Immobile behavior group, and if $u$ is in the Mobile behavior group, the reconfiguration of $u$ must be locally connected (Definition 11). We show the following lemma:

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph and let $G' = (V, E')$ be the graph resulting from a valid locally connnected reconfiguration of a vertex $u \in V$. If a configuration satisfies the state invariant on $G$, then the same state assignments satisfy the state invariant on $G'$.*

**Proof.** As locally connected reconfigurations of Unaware agents do not affect the invariant and Immobile agents cannot be reconfigured, it suffices to show that locally connected reconfigurations of Mobile agents maintain the state invariant.

To show that the state invariant holds on $G'$, we show that any Mobile agent has a path to an Immobile agent in $G'$. Consider any such Mobile agent $v \neq u$. As the state invariant is satisfied on $G$, there exists a path in $G$ over Aware vertices from $v$ to an Immobile agent $w$. If this path does not contain the agent $u$, $v$ has a path to $w$ in $G'$. On the other hand, if this path contains the agent $u$, consider the vertices $u_1$ and $u_2$ before and after $u$ respectively in this path. By local connectivity, there must still be a path from $u_1$ to $u_2$ in the induced subgraph $G'[N_{\mathcal{A}}(u) \cup \{u\}]$, and so a path exists from $v$ to $w$ over $G'$. It remains to check that $u$ also has a path to an Immobile agent in $G'$. Once again by local connectivity, $u$ must have an Aware neighbor in $G'$, which must have a path to an Immobile agent over $G'$. ◀

In the reconfigurable version of the dynamic stimuli problem, we have a random sequence of graphs $(G_0, G_1, G_2, \ldots)$ where $G_t$ for $t \geq 1$ denotes the graph used in iteration $t$. These graphs share a common vertex set $V$, the set of agents, but the edges may change from iteration to iteration. We do not consider fully arbitrary sequence of graphs, but instead one that is generated by what we call a *(valid) reconfiguration adversary* $\mathcal{X}$. Let the random sequence $(X_0, X_1, X_2, \ldots)$ denote the information available to the reconfiguration adversary on each iteration, and for each $t \geq 1$ we let the vector $\widehat{\sigma}_t : V \to \{\widehat{\mathcal{U}}, \widehat{\mathcal{M}}, \widehat{\mathcal{I}}\}$ denote the behavior groups of the agents at the end of iteration $t$ (i.e., after the activated agent performed any computation/change of states of its neighborhood at iteration $t$). At iteration $t$, before an agent is activated in the stimuli algorithm, the new graph $G_t$ and the next value $X_t$ of the

sequence are drawn as a pair from the distribution $\mathcal{X}(X_{t-1}, \widehat{\sigma}_{t-1})$, which assigns non-zero probabilities only to graphs that can be obtained through some sequence of valid locally connected reconfigurations of the vertices of $G_{t-1}$.

We note that the reconfiguration adversary can be deterministic or randomized (it can even be in control of the agents themselves), and is specifically defined to act based on the behavior group vectors $\widehat{\sigma}_t : V \to \{\widehat{\mathcal{U}}, \widehat{\mathcal{M}}, \widehat{\mathcal{I}}\}$ and *not* on the state vector of the agents. We explicitly do not give the reconfiguration adversary access to full state information, as convergence time bounds require that the reconfiguration adversary of the graph be oblivious to the movements of alert tokens. As a special case, our results hold for any sequence of graphs $(G_0, G_1, G_2, \ldots)$ pre-determined by an oblivious adversary. An example of a valid reconfiguration adversary that takes full advantage of the generality of our definition can be seen in the Adaptive $\alpha$-Compression Algorithm, an algorithm that we will later introduce to solve the problem of foraging.

In the static version of the problem, the graph is required to be connected to ensure that agents will always be able to communicate with each other. Without this requirement, we can imagine simple examples of graphs or graph sequences where no algorithm will work. In particular, if a set of agents that contains a witness never forms an edge to an agent outside of the set, there would be no way to transmit information about the existence of the witnesses to the nodes outside the set. However, as the foraging problem will require disconnections to some extent, we relax this requirement that each graph $G_t$ is connected, and instead quantify how frequently UNAWARE state agents come into contact with AWARE state agents.

We say an UNAWARE agent is *active* if it is adjacent to an AWARE agent. One way to quantify how frequently agents become active is to divide the iterations into "batches" of bounded expected duration, with at least some amount of active agents in each batch. The random variables $(D_1, D_2, D_3 \ldots)$ denote the durations (in iterations) of these batches, and the random variables $(C_1, C_2, C_3, \ldots)$ denote the number of active agents in the respective batches. Definition 13 formalizes this notion.

▶ **Definition 13** (Recurring Sequences). *Let $\mathcal{X}$ be a fixed valid reconfiguration adversary. We say that this $\mathcal{X}$ is* $(U_D, U_C)$-recurring *(for $U_D \geq 1$ and $U_C \in (0,1)$) if for each possible starting iteration $t$ and fixed behavior group $\widehat{\sigma}$ with at least one* UNAWARE *and one* IMMOBILE *agent, we have the following property: There exists sequences of random variables $(D_1, D_2, D_3 \ldots)$ and $(C_1, C_2, C_3 \ldots)$ where for each $k \in \{1, 2, 3, \ldots\}$,*

1. *$C_k$ denotes the number of active agents under the behavior group $\widehat{\sigma}$ between iterations $t + \sum_{i=1}^{k-1} D_i$ and $t + \left(\sum_{i=1}^{k} D_i\right) - 1$.*
2. *$\mathbb{E}\left[D_k \mid D_1, D_2, \ldots D_{k-1}, C_1, C_2 \ldots C_{k-1}\right] \leq U_D$.*
3. *$\mathbb{E}\left[\left(1 - \frac{1}{n}\right)^{C_k} \mid D_1, D_2, \ldots D_{k-1}, C_1, C_2 \ldots C_{k-1}\right] \leq U_C$.*

We can then define a *(valid) reconfigurable graph (or sequence)* as one that is generated by a valid reconfiguration adversary $\mathcal{X}$ and is $(U_D, U_C)$-recurring for some $U_D \geq 1, U_C \in (0,1)$. This allows us to state our main results for reconfigurable graphs as Theorems 14 and 15. The theorems we have shown for the static version of the problem (Theorems 3 and 4) are special cases of these two theorems.

▶ **Theorem 14.** *Starting from any configuration satisfying the state invariant over a reconfigurable graph, if $|\mathcal{W}_T| = 0$, then all agents will reach and remain in the* UNAWARE *state in $O(n^2)$ expected iterations, after time $T$.*

▶ **Theorem 15.** *Starting from any configuration satisfying the state invariant over a reconfigurable graph, if $|\mathcal{W}_T| > 0$ and the reconfiguration adversary is $(U_D, U_C)$-recurring, then all agents will reach and stay in the* Aware *state in $O\left(n^6 \log n + \frac{nU_D}{1-U_C}\right)$ expected iterations, after time $T$.*

In particular, if every graph $G_t$ is connected, then the reconfiguration adversary is $\left(1, \left(1 - \frac{1}{n}\right)\right)$-recurring by setting $D_k = C_k = 1$ (as constant random variables) for all $k$, and we have the following corollary:

▶ **Corollary 16.** *Starting from any configuration satisfying the state invariant over a reconfigurable graph, if $|\mathcal{W}_T| > 0$ and every $G_t$ is connected, then all agents will reach and stay in the* Aware *state in $O(n^6 \log n)$ expected iterations, after time $T$.*

Obviously, if $G$ is a static connected graph, it falls as a special case of the corollary; a tighter analysis allowed us to prove the $O(n^5)$ expected convergence bound in Theorem 4.

The following two lemmas, which are analogous to Lemmas 8 and 9 but for reconfigurable graphs, are sufficient to show Theorem 14 (the case for $|\mathcal{W}_T| = 0$). The proof of Lemma 17 is identical to the proof of Lemma 8, so we only show Lemma 18.

▶ **Lemma 17.** *We start from a configuration satisfying the state invariant over a reconfigurable graph with no more than $w$ witnesses at any point. Then the expected number of steps before we reach a configuration with no residual components is at most $2n^2/(1 - wp)$.*

▶ **Lemma 18.** *We start from a configuration satisfying the state invariant over a reconfigurable graph, and assume that no agent will be removed from the witness set from the current point on. If there are currently no residuals, then a residual cannot be generated.*

**Proof.** From the proof of Lemma 9, we know that state changes of agents do not generate a new residual. Valid reconfigurations of agents also cannot generate a new residual, as from Lemma 12, the state invariant always holds, so all components will always have an agent in state $\mathcal{A}_{\{C\}}$, $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$. Reconfigurations cannot change the fact that there will be no agent of state $\mathcal{A}_{\{C\}}$, and that all agents in states $\mathcal{A}_{\{W\}}$ or $\mathcal{A}_{\{A,W\}}$ will be witnesses. ◀

The key result we will show is Lemma 19, a loose polynomial-time upper bound for the amount of time before the next agent switches to the Aware state. This gives a polynomial time bound for all agents switching to the Aware state when $|\mathcal{W}_T| > 0$, implying Theorem 15.

▶ **Lemma 19.** *We start from a configuration satisfying the state invariant over a $(U_D, U_C)$-recurring reconfigurable graph, and assume that there are no residuals, the witness set is nonempty, and no agent will be removed from the witness set from the current point on. Then the expected number of iterations before the next agent switches from the* Unaware *to the* Aware *state is at most $O\left(n^5 \log n + \frac{U_D}{1-U_C}\right)$.*

**Proof Sketch.** This proof largely follows the proof of Lemma 10, with some modifications to allow for reconfigurability. In the interest of space however, we will provide only a summary of the key ideas and calculations that go into the proof (which is available in the full version of the paper). Assuming that every witness is already in the Aware state with the witness flag set, we upper bound the expected time it takes for all Aware agents to obtain an alert token, followed by the time it takes for an agent to be activated while adjacent to an alert token and switch to the Aware state.

To bound the expected time for all AWARE agents to obtain an alert token, we apply the same strategy, marking an agent without an alert token and passing around the mark until it lands on a witness. We make use of the result of [3, 13], which states that the expected hitting time of the $d_{max}$-random walk on a connected evolving graph controlled by an *oblivious adversary* is $O(n^3 \log n)$ [13]. This corresponds to $O(n^4 \log n)$ iterations in expectation to generate a new alert token, which gives an upper bound of $O(n^5 \log n)$ iterations in expectation before all agents carry alert tokens.

Two complications arise however when applying this result - the requirement for the adversary controlling the dynamic graph to be oblivious and the requirement that the dynamic graph remains connected. The first issue is dealt with with an observation that with no change in the behavior group vector (as long as no new agent switches to the AWARE state), the sequence of graphs generated by agent movement is independent of the movement of alert tokens. The second issue is resolved with the observation that even though each graph $G_t[A]$ induced by the set of AWARE agents is not connected, the state invariant and the lack of residuals ensure that each of its connect components contains a witness. The witnesses are linked with imaginary edges to connect the graph, which we can do as we only care about the amount of time before the mark lands on any witness.

A new AWARE agent is added when an active UNAWARE agent is activated. The probability of adding a new AWARE agent on a given iteration with $k$ active agents is thus $\frac{k}{n}$, as each of the $n$ agents are activated with equal probability. Thus, if we denote by the random sequence $K_1, K_2, K_3, \ldots$ the number of active agents on each iteration following $T_{\text{full}}$ (including $T_{\text{full}}$), we get the following expression for the expected value of $X$, which we use to denote the number of iterations following $T_{\text{full}}$ before a new AWARE agent is added:

$$\begin{aligned}
&\mathbb{E}[X|K_1, K_2, K_3, \ldots] = \sum_{x=0}^{\infty} Pr\left(X > x | K_1, K_2, K_3, \ldots\right) \\
&= 1 + \sum_{x=1}^{\infty} \prod_{i=1}^{x} \left(1 - \frac{K_i}{n}\right) \leq 1 + \sum_{x=1}^{\infty} y^{\sum_{i=1}^{x} K_i} \text{ where } y := \left(1 - \frac{1}{n}\right) \in (0,1) \\
&= 1 + \underbrace{y^{K_1} + y^{K_1+K_2} + \ldots + y^{\sum_{i=1}^{D_1} K_i}}_{D_1 \text{ terms}} + \underbrace{y^{C_1+K_{D_1+1}} + \ldots + y^{C_1+\sum_{i=D_1+1}^{D_1+D_2} K_i}}_{D_2 \text{ terms}} + \ldots \\
&\leq D_1 + D_2 \cdot y^{C_1} + D_3 \cdot y^{C_1+C_2} + \ldots \text{ (as } \sum_{i=1}^{D_1+D_2+\ldots+D_x} K_i = C_{x+1} \text{ for all } x).
\end{aligned}$$

Thus, via the law of total expectation, we have

$$\begin{aligned}
\mathbb{E}[X] \leq \sum_{i=1}^{\infty} \mathbb{E}[D_i y^{\sum_{j=1}^{i-1} C_j}] &\leq \sum_{i=1}^{\infty} \mathbb{E}\left[y^{\sum_{j=1}^{i-1} C_j} \mathbb{E}[D_i | C_1, C_2, \ldots, C_{i-1}]\right] \\
&\leq \sum_{i=1}^{\infty} U_D \mathbb{E}\left[y^{\sum_{j=1}^{i-2} C_j} \mathbb{E}[y^{C_{i-1}} | C_1, C_2, \ldots, C_{i-2}]\right] \\
&\leq \ldots \leq \sum_{i=1}^{\infty} U_D (U_C)^{i-1} = \frac{U_D}{1 - U_C}.
\end{aligned}$$

This gives us an expected time bound of $O\left(n^5 \log n + \frac{U_D}{1-U_C}\right)$ to add a new AWARE agent. ◀

## 5 Foraging via self-induced phase changes

Recall that in *the foraging problem*, we have "ants" (agents) that may initially be searching for "food" (stimuli, which can be any resource in the environment, like an energy source); once a food source is found, the ants that have learned about the food source start informing other ants, allowing them to switch their behaviors from the *search mode* to the *gather mode*, that leads them to start to gather around the food source to consume the food. Once the source is depleted, the ants closer to the depleted source start a broadcast wave, gradually informing other ants that they should restart the search phase again by individually switching their states. The foraging problem is very general and has several fundamental application domains, including search-and-rescue operations in swarms of nano- or micro-robots; health applications (e.g., a collective of nano-sensors that could search for, identify, and gather around a foreign body to isolate or consume it, then resume searching, etc.); and finding and consuming/deactivating hazards in a nuclear reactor or a minefield.

Our model for foraging is based on the *geometric Amoebot model* for programmable matter [11, 12]. We have $n$ anonymous agents occupying distinct sites on a $\sqrt{N} \times \sqrt{N}$ piece of the triangular lattice with periodic boundary conditions. These agents have constant-size memory, and have no global orientation or any other global information beyond a common chirality. Agents are activated with individual Poisson clocks, upon which they may move to adjacent unoccupied sites or change states, operating under similar constraints to the dynamic stimuli problem. An agent may only communicate with agents occupying adjacent sites of the lattice. Food sources may be placed on any site of the lattice, removed, or shifted around at arbitrary times, possibly adversarially, and an agent can only observe the presence of the food source while occupying the lattice site containing it. This model can be viewed as a high level abstraction of the (canonical) *Amoebot model* [11, 12] under a *random sequential scheduler*, where at most one agent would be active at any point in time. One should be able to port the model and algorithms presented in this paper to the Amoebot model; however a formal description on how this should be done is beyond the scope of this paper.

At any point of time, there are two main states an agent can be in, which, at the macro-level, are to induce the collective to enter the *search* or *gather* modes respectively. When in *search mode*, agents move around in a process akin to a simple exclusion process, where they perform a random walk while avoiding two agents occupying the same site. Agents enter the *gather mode* when food is found and this information is propagated in the system, consequently resulting in the system compressing around the food (Figure 5).

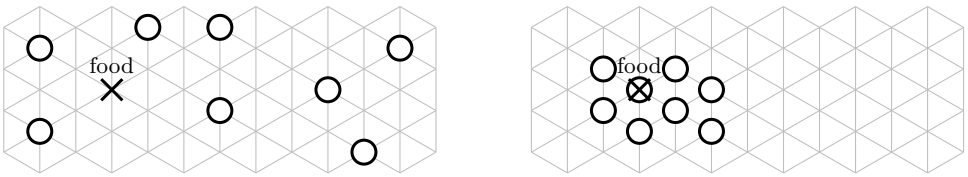



**Figure 5** In the diagram on the left, a food source is placed on a lattice site. The diagram on the right illustrates a desired configuration, where all agents have gathered in a low perimeter configuration around the food source. If the food source is later removed, the agents should once again disperse, returning to a configuration like the figure on the left.

In the nonadaptive setting, Cannon *et al.* [5] designed a rigorous compression/expansion algorithm for agents that remain simply connected throughout execution, where a single parameter $\lambda$ determines a system-wide phase: A small $\lambda$, namely $\lambda < 2.17$, provably corresponds to the search mode, which is desirable to search for food, while large $\lambda$, namely

$\lambda > 2 + \sqrt{2}$, corresponds to the gather mode, desirable when food has been discovered. Likewise, Li *et al.* [25] show a very similar bifurcation based on a bias parameter $\lambda$ in the setting when the agents are allowed to disconnect and disperse throughout the lattice. Our goal here is to perform a system-wide adjustment in the bias parameters when one or more agents notice the presence or depletion of food to induce the appropriate global coordination to provably transition the collective between macro-modes when required. Informally, one can imagine individual agents adjusting their $\lambda$ parameter to be high when they are fed, encouraging compression around the food, and making $\lambda$ small when they are hungry, promoting the search for more food. A configuration is called *$\alpha$-compressed* if the perimeter (measured by the length of the closed walk around its boundary edges) is at most $\alpha\, p_{\min}(n)$, for some constant $\alpha > 1$, where $p_{\min}(n)$ denotes the minimum possible perimeter of a connected system with $n$ agents, which is the desired outcome of the gather mode.

**Adaptive $\alpha$-compression.** We present the first rigorous local distributed algorithm for the foraging problem: The *Adaptive $\alpha$-Compression* algorithm is based on the stochastic compression algorithm of [5], addressing a geometric application of the dynamic stimuli problem, where the Aware state represents the "gather" mode, and the Unaware state represents the "search" mode. A witness is an agent that occupies the same lattice site as the food source. The underlying dynamic graph used by the Adaptive Stimuli Algorithm is given by the adjacency graph of the agents - two agents share an edge on the graph if they occupy adjacent sites of the lattice. As agents move around to implement behaviors like gathering and searching, their neighbor sets will change. The movement of agents thus reconfigures and oftentimes even disconnects our graph.

In this algorithm, agents in the Unaware (search) behavior group execute movements akin to a simple exclusion process (Execute-Search) while agents in the Mobile (gather) behavior group execute moves of the compression algorithm (Execute-Gather) in [5]. We focus on the compression algorithm run by the Aware agents. In a simple exclusion process, a selected agent picks a direction at random, and moves in that direction if and only if the immediate neighboring site in that direction is unoccupied. In the compression algorithm [5] on the other hand, a selected Aware agent first picks a direction at random to move in, and if this move is a valid compression move (according to Definition 20), the agent moves to the chosen position with the probability given in Definition 21. With a (far-from-trivial) modification of the analysis in [5] to account for the stationary witness agent, we show that in the case of a single food source, the "gather" movements allow the agents to form a low perimeter cluster around the food. We present the following definitions, adapted from [5] to the set of Aware agents running the compression algorithm:

▶ **Definition 20** (Valid Compression Moves [5])**.** *Denote by $N_{\mathcal{A}}(\ell)$ and $N_{\mathcal{A}}(\ell')$ the sets of Aware neighbors of $\ell$ and $\ell'$ respectively and $N_{\mathcal{A}}(\ell \cup \ell') := N_{\mathcal{A}}(\ell) \cup N_{\mathcal{A}}(\ell') \setminus \{\ell, \ell'\}$. Consider the following two properties:*

*Property 1: $|N_{\mathcal{A}}(\ell) \cap N_{\mathcal{A}}(\ell')| \geq 1$ and every agent in $N_{\mathcal{A}}(\ell \cup \ell')$ is connected to an agent in $N_{\mathcal{A}}(\ell) \cap N_{\mathcal{A}}(\ell')$ through $N_{\mathcal{A}}(\ell \cup \ell')$.*

*Property 2: $|N_{\mathcal{A}}(\ell) \cap N_{\mathcal{A}}(\ell')| = 0$, $\ell$ and $\ell'$ each have at least one neighbor, all agents in $N_{\mathcal{A}}(\ell) \setminus \{\ell'\}$ are connected by paths within the set, and all agents in $N_{\mathcal{A}}(\ell') \setminus \{\ell\}$ are connected by paths within the set.*

*We say the move from $\ell$ to $\ell'$ is a* valid compression move *if it satisfies both properties, and $N_{\mathcal{A}}(\ell)$ contains fewer than five aware state agents.*

**Algorithm 2** Adaptive $\alpha$-Compression.

**procedure** ADAPTIVE-ALPHA-COMPRESSION($u$)
  $q \leftarrow$ Random number in $[0, 1]$
  $u.isWitness \leftarrow$TRUE if $u$ observes the food source, else $u.isWitness \leftarrow$FALSE
  **if** $q \leq \frac{1}{2}$ **then** ▷ With probability $\frac{1}{2}$, make a state update
    ADAPTIVE-STIMULI-ALGORITHM(u)
  **else** ▷ With probability $\frac{1}{2}$, make a move
    **if** $u.isWitness$ or $u.state \in \{\mathcal{A}_{\{W\}}, \mathcal{A}_{\{A,W\}}, \mathcal{A}_{\{C\}}\}$ **then** ▷ IMMOBILE agent
      Do nothing
    **else if** $u.state \in \{\mathcal{A}_{\emptyset}, \mathcal{A}_{\{A\}}\}$ **then** ▷ MOBILE agent
      EXECUTE-GATHER(u)
    **else if** $u.state = \mathcal{U}$ **then** ▷ UNAWARE agent
      EXECUTE-SEARCH(u)

**procedure** EXECUTE-GATHER($u$)
  $d \leftarrow$ Random direction in $\{0, 1, 2, 3, 4, 5\}$
  $\ell \leftarrow$ Current position of $u$
  $\ell' \leftarrow$ Neighboring lattice site of $u$ in direction $d$
  **if** Moving $u$ from $\ell$ to $\ell'$ is a valid compression move (Definition 20) **then**
    $p \leftarrow$ Random number in $[0, 1]$
    $d(u) \leftarrow$ number of neighboring agents of $u$ if $u$ were at position $\ell$
    $d'(u) \leftarrow$ number of neighboring agents of $u$ if $u$ were at position $\ell'$
    **if** $p \leq \lambda^{d'(u)-d(u)}$ **then**
      Move $u$ to position $\ell'$ ▷ Movements reconfigure the adjacency graph

**procedure** EXECUTE-SEARCH($u$)
  $d \leftarrow$ Random direction in $\{0, 1, 2, 3, 4, 5\}$
  $\ell' \leftarrow$ Neighboring lattice site of $u$ in direction $d$
  **if** Moving $\ell'$ is an unoccupied lattice site **then**
    Move $u$ to position $\ell'$ ▷ Movements reconfigure the adjacency graph

▶ **Definition 21** (Transition probabilities [5]). *Fix $\lambda > 2 + \sqrt{2}$, as sufficient for $\alpha$-compression. An agent $u$ transitions through a valid movement with Metropolis-Hastings [28] acceptance probability $\min\{1, \lambda^{e(\sigma')-e(\sigma)}\}$, where $\sigma$ and $\sigma'$ are the configurations before and after the movement, and $e(\cdot)$ represents the number of edges between AWARE state agents in the configuration.*

Note that even though $e(\cdot)$ is a global property, the difference $e(\sigma') - e(\sigma)$ can be computed locally (within two hops in the lattice, or through expansions in the Amoebot model [11, 12]), as it is just the change in the number of AWARE neighbors of $u$ before and after its movement.

The condition for valid compression moves is notable as it keeps a component of aware agents containing the witness agent simply connected (connected and hole-free), which is crucial to the proof in [5] that a low perimeter configuration, in our case around the food source, will be obtained in the long term. We also show that these valid compression moves are locally connected (as per Definition 11), a sufficient condition for the reconfigurable dynamic stimuli problem to apply.

We first prove that the Markov chain representing the compression moves (EXECUTE-GATHER) is connected. The proof builds upon the ergodicity argument in [5]; however, the addition of a single stationary agent, the witness, in our context makes this proof significantly more complex, and we defer it to the full version of the paper.

▶ **Lemma 22.** *Consider connected configurations of agents on a triangular lattice with a single agent $v$ that cannot move. There exists a sequence of valid compression moves that transforms any connected configuration of agents into any simply connected configuration of the agents while keeping $v$ stationary.*

We may now state our main results, which verify the correctness of Adaptive $\alpha$-Compression.

▶ **Theorem 23.** *If no food source has been identified for sufficiently long, then within an expected $O(n^2)$ steps, all agents will reach and remain in the* Unaware *state and will converge to the uniform distribution of nonoverlapping lattice positions.*

▶ **Theorem 24.** *If at least one food source exists and remains in place for long enough, then within $O(n^6 \log n + N^2 n)$ steps in expectation, all agents will reach and remain in the* Aware *state, and each component of* Aware *agents will contain a food source. In addition, if there is only one food source, the agents will converge to a configuration with a single $\alpha$-compressed component around the food, for any constant $\alpha > 1$, with all but an exponentially small probability, for a large enough lattice region.*

The Adaptive $\alpha$-Compression Algorithm fits the requirements of the reconfigurable dynamic stimuli model. In particular, the information $X_t$ available to the reconfiguration adversary corresponds to the configuration of the lattice, and the graph $G_t$ represents the adjacency of agents on the lattice at that time $t$. To show that a sequence of valid Aware agent movements in the Adaptive $\alpha$-Compression Algorithm, which determine the configurations of $G_1, G_2, \ldots$, can be modeled via a valid reconfiguration adversary $\mathcal{X}$, we need to show that the reconfigurations resulting from the Execute-Gather procedure must be locally connected.

▶ **Lemma 25.** *The movement behavior of Adaptive $\alpha$-Compression is locally connected.*

**Proof.** We only need to show that the Execute-Gather procedure maintains local connectivity (Definition 11). This is true as when reconfiguring an Aware agent $u$ with Aware neighbor set $N_{\mathcal{A}}(u)$, we only allow valid compression moves (Definition 20) to be made. In the case of Property 1, all agents in $N_{\mathcal{A}}(u)$ will still have paths to $u$ in $G'$ through $S$. In the case of Property 2, all agents in $N_{\mathcal{A}}(u)$ will still have paths to each other within $G'[N_{\mathcal{A}}(u)]$, despite no longer having local paths to $u$. The agent $u$ will have at least one Aware state neighbor after the move as this is a requirement of Property 2. ◀

As this is an instance of the reconfigurable dynamic stimuli problem, Theorem 23 follows immediately from Theorem 14. To show Theorem 24 however, we need to show polynomial recurring rates by arguing that the Unaware state agents following a simple exclusion process will regularly come into contact with the clusters of Aware state agents around the food sources.

▶ **Lemma 26.** *The movement behavior defined in the Adaptive $\alpha$-Compression Algorithm is $(U_D, U_C)$-recurring with $U_D = 2N^2 + \frac{2}{n} + 1$ and $U_C = \frac{2}{3}$.*

**Proof Sketch.** In the interest of space, we give only a brief summary of the proof (which is available in the full version of the paper). We define the random sequences $(D_1, D_2, D_3, \ldots)$ and $(C_1, C_2, C_3, \ldots)$ by dividing the time steps after the starting iteration $t$ into batches, where the $k^{\text{th}}$ batch would take $D_k$ iterations and would see $C_k$ active agents over its duration.

A batch ends (and the next batch starts) when the agent movement places an UNAWARE agent $u$ next to an AWARE agent $v$, then attempts a movement of $u$ or $v$ after that. The duration $D_k$ of the $k^{\text{th}}$ batch can be computed with the hitting time of a simple exclusion process over the triangular lattice, plus a geometric random variable representing the number of iterations taken to select $u$ or $v$ after that. This gives us a uniform upper bound of $2N^2n + \frac{2}{n} + 1$ for $\mathbb{E}[D_k | D_1, D_2, \ldots D_{k-1}, C_1, C_2 \ldots C_{k-1}]$ for each batch $k \in \{1, 2, \ldots\}$.

The number of active agents $C_k$ within batch $k$ is at least the number of iterations between the first time within the batch that an UNAWARE agent moves next to an AWARE agent and the end of the batch. This is shown to stochastically dominate a geometric random variable $Y$ with success probability $p_Y$, which we show in the full paper to be:

$$p_Y = \sum_{i=0}^{\infty} \frac{1}{2} \cdot \left(\frac{1}{2}\right)^i \left(1 - \left(1 - \frac{2}{n}\right)^i\right) = 1 - \frac{1}{2} \sum_{i=0}^{\infty} \left(\frac{1 - 2/n}{2}\right)^i = \frac{2}{n} \left(\frac{1}{1 + 2/n}\right)$$

As $C_k$ stochastically dominates $Y$ and $(1 - \frac{1}{n})^x$ is a decreasing function of $x$, we have:

$$\begin{aligned} \mathbb{E}\left[\left(1 - \frac{1}{n}\right)^{C_k} | D_1, D_2, \ldots D_{k-1}, C_1, C_2 \ldots C_{k-1}\right] &\leq \mathbb{E}\left[\left(1 - \frac{p}{n}\right)^Y\right] \\ &= \sum_{y=0}^{\infty} \left(1 - \frac{1}{n}\right)^y p_Y \left(1 - p_Y\right)^y \\ &= p_Y \frac{1}{1 - (1 - 1/n)(1 - p_Y)} = \frac{2}{3}. \end{aligned}$$

◀

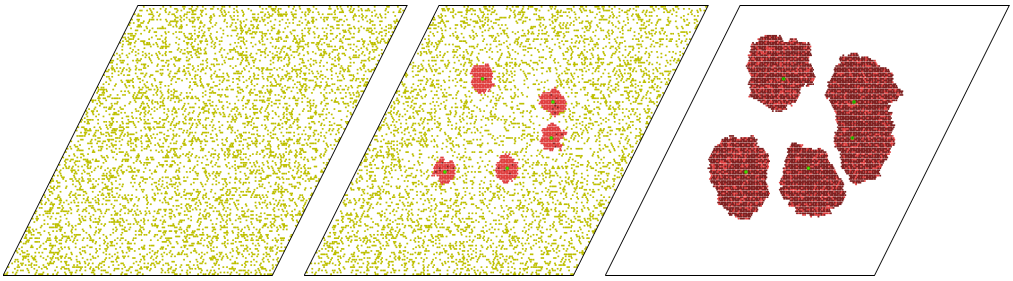
To show the first half of Theorem 24, we start from the first iteration beyond which no additional changes in the positions (or existence) of the food sources occur. We first show that if there is at least one food source, it will be found. As long as no food source has been found, there will be no witnesses, so every agent will return to the UNAWARE state by Theorem 14. Agents in the UNAWARE state move randomly following a simple exclusion process. Using the hitting time of a simple random walk on a regular graph (the triangular lattice) of $N$ sites, we have a simple upper bound of $O(N^2 n)$ iterations in expectation before some agent finds a food source and becomes a witness.

From then on, there will be at least one witness, and the witness set can only be augmented, not reduced, as the other agents potentially find additional food sources, and since the agents already sitting on food sources are no longer allowed to move. As agent movement behaviors are recurring with polynomial bounds (Lemma 26), the reconfigurable Adaptive Stimuli Algorithm applies, yielding a polynomial bound on the expected number of iterations before all agents have switched to the AWARE state with no residuals. Additionally, due to the maintenance of the state invariant and as there are no residuals, every component of AWARE agents will contain at least one witness, meaning that every cluster of agents will be around some food source. This gives us the first part of Theorem 24.

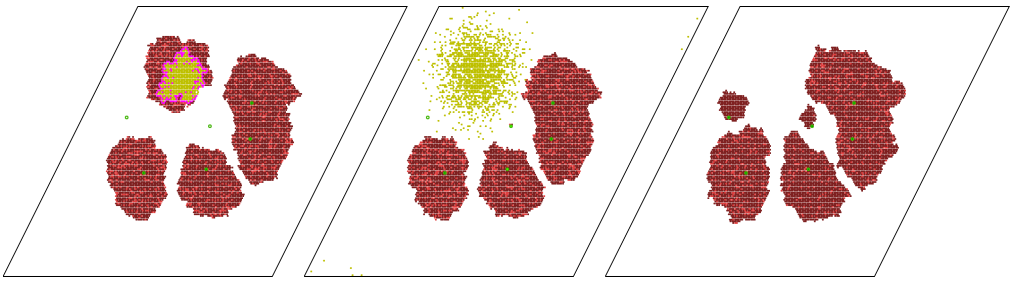
The second half of Theorem 24 states that a low perimeter ($\alpha$-compressed) configuration is achievable in the case of a single food source. As the Markov chain representing the compression moves is irreducible (Lemma 22), the results of [5] guarantee that for any $\alpha > 1$, there exists a sufficiently large constant $\lambda$ such that at stationarity, the perimeter of the cluster is at most $\alpha$ times its minimum possible perimeter with high probability.

## 6 Simulations of the Adaptive $\alpha$-Compression Algorithm

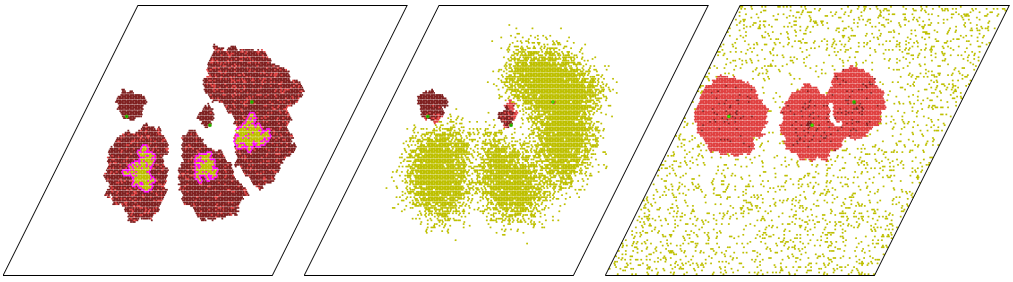
We demonstrate a simulation of the Adaptive $\alpha$-Compression algorithm with 5625 agents in a $150 \times 150$ triangular lattice with periodic boundary conditions. Multiple food sources (stimuli) are placed and moved around to illustrate the gather and search phases. This simulation is shown as a sequence of 12 images in chronological order in Figure 6.



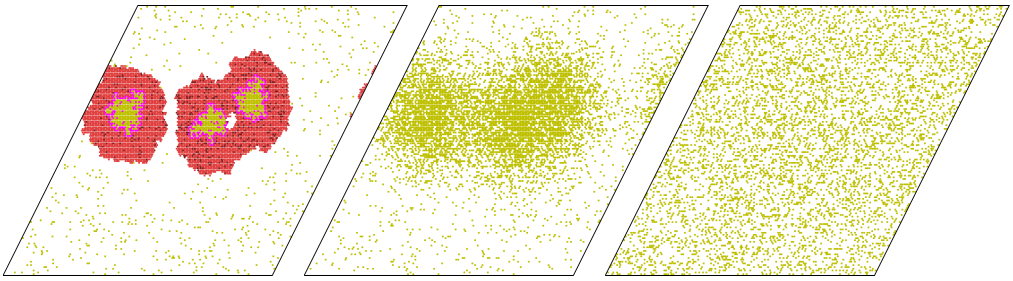
**(a)** All agents initially unaware. Five food sources added, all agents eventually switch to the AWARE state.



**(b)** One food source removed, two new added. A cluster disperses and agents rejoin other clusters.



**(c)** Three food sources removed, clusters disperse and gather around the remaining food sources.



**(d)** All food sources removed, agents disperse and converge to a uniform distribution over the lattice.

**Figure 6** Simulation of Adaptive $\alpha$-Compression with multiple food sources. The images are in chronological order. UNAWARE agents are yellow, AWARE agents are red (darker red if they have an alert token), agents with the all-clear token are purple, and food sources are green.

## References


1 Simon Alberti. Organizing living matter: The role of phase transitions in cell biology and disease. *Biophysical journal*, 14, 2018.

2 Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A stochastic approach to shortcut bridging in programmable matter. In *23rd International Con- ference on DNA Computing and Molecular Programming (DNA)*, pages 122–138, 2017.

3 Chen Avin, Michal Koucký, and Zvi Lotker. Cover time and mixing time of random walks on dynamic graphs. *Random Structures & Algorithms*, 52(4):576–596, 2018.

4 Sarah Cannon, Joshua J. Daymude, Cem Gökmen, Dana Randall, and Andréa W. Richa. A local stochastic algorithm for separation in heterogeneous self-organizing particle systems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, pages 54:1–54:22, 2019.

5 Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A Markov chain algorithm for compression in self-organizing particle systems. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 279–288, 2016.

6 Arnaud Casteigts. Finding structure in dynamic networks, 2018. `arXiv:1807.07801`.

7 Bernard Chazelle. The convergence of bird flocking. *J. ACM*, 61(4), 2014.

8 Andrea Clementi, Riccardo Silvestri, and Luca Trevisan. Information spreading in dynamic graphs. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, pages 37–46, 2012.

9 Nikolaus Correll and Alcherio Martinoli. Modeling and designing self-organized aggregation in a swarm of miniature robots. *The Int'l J. of Robotics Research*, 30(5):615–626, 2011.

10 Paolo Dario, Renzo Valleggi, Maria Chiara Carrozza, M. C. Montesi, and Michele Cocco. Microactuators for microrobots: a critical survey. *Journal of Micromechanics and Microengineering*, 2(3):141–157, 1992.

11 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The Canonical Amoebot Model: Algorithms and Concurrency Control. In *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, 2021.

12 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. *Distributed Computing*, 2023. To appear.

13 Oksana Denysyuk and Luís Rodrigues. Random walks on evolving graphs with recurring topologies. In *Distributed Computing*, pages 333–345. Springer Berlin Heidelberg, 2014.

14 Michael Dinitz, Jeremy T. Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of information spreading in dynamic networks. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC)*, volume 246 of *LIPIcs*, pages 18:1–18:22, 2022.

15 Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. On the complexity of information spreading in dynamic networks. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 717–736. SIAM, 2013.

16 Nazim Fatès. Solving the decentralised gathering problem with a reaction–diffusion–chemotaxis scheme. *Swarm Intelligence*, 4(2):91–115, 2010.

17 Nazim Fatès and Nikolaos Vlassopoulos. A robust aggregation method for quasi-blind robots in an active environment. In *ICSI 2011*, 2011.

18 Simon Garnier, Jacques Gautrais, Masoud Asadpour, Christian Jost, and Guy Theraulaz. Self-organized aggregation triggers collective decision making in a group of cockroach-like robots. *Adaptive Behavior*, 17(2):109–133, 2009.

19 Simon Garnier, Christian Jost, Raphaël Jeanson, Jacques Gautrais, Masoud Asadpour, Gilles Caprari, and Guy Theraulaz. Aggregation behaviour as a source of collective decision in a group of cockroach-like-robots. In *Advances in Artificial Life*, ECAL '05, pages 169–178, 2005.

20 Bernhard Haeupler and David Karger. Faster information dissemination in dynamic networks via network coding. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 381–390, 2011.

21 Walter Hussak and Amitabh Trehan. On termination of a flooding process. In *Proc. of the 2019 ACM Symp. on Principles of Distributed Computing*, PODC '19, pages 153–155, 2019.

22 Hridesh Kedia, Shunhao Oh, and Dana Randall. A local stochastic algorithm for alignment in self-organizing particle systems. In *Approximation, Randomization, and Combinatorial*

*Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*, volume 245, pages 14:1–14:20, 2022.

23 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, pages 513–522, 2010.

24 Fabian Kuhn and Rotem Oshman. Dynamic networks: Models and algorithms. *SIGACT News*, 42(1):82–96, 2011.

25 Shengkai Li, Bahnisikha Dutta, Sarah Cannon, Joshua J. Daymude, Ram Avinery, Enes Aydin, Andréa W. Richa, Daniel I. Goldman, and Dana Randall. Programming active granular matter with mechanically induced phase changes. *Science Advances*, 7, 2021.

26 Jintao Liu, Arthur Prindle, Jacqueline Humphries, Marçal Gabalda-Sagarra, Munehiro Asally, Dong-Yeon D. Lee, San Ly, Jordi Garcia-Ojalvo, and Gürol M. Süel. Metabolic co-dependence gives rise to collective oscillations within biofilms. *Nature*, 523(7562):550–554, 2015.

27 Anne E. Magurran. The adaptive significance of schooling as an anti-predator defence in fish. *Annales Zoologici Fennici*, 27(2):51–66, 1990.

28 Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

29 Nathan J. Mlot, Craig A. Tovey, and David L. Hu. Fire ants self-assemble into waterproof rafts to survive floods. *Proc. of the National Academy of Sciences*, 108(19):7669–7673, 2011.

30 Anil Özdemir, Melvin Gauci, Salomé Bonnet, and Roderich Groß. Finding consensus without computation. *IEEE Robotics and Automation Letters*, 3(3):1346–1353, 2018.

31 Arthur Prindle, Jintao Liu, Munehiro Asally, San Ly, Jordi Garcia-Ojalvo, and Gürol M. Süel. Ion channels enable electrical communication in bacterial communities. *Nature*, 527(7576):59–63, 2015.

32 Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, pages 10–20, 2005.

33 William Savoie, Sarah Cannon, Joshua J. Daymude, Ross Warkentin, Shengkai Li, Andréa W. Richa, Dana Randall, and Daniel I. Goldman. Phototactic supersmarticles. *Artificial Life and Robotics*, 23(4):459–468, 2018.

34 Thomas C. Schelling. Dynamic models of segregation. *The Journal of Mathematical Sociology*, 1(2):143–186, 1971.

35 Onur Soysal and Erol Şahin. Probabilistic aggregation strategies in swarm robotic systems. In *Proceedings 2005 IEEE Swarm Intelligence Symposium*, SIS 2005, pages 325–332, 2005.

36 Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Physica D: Nonlinear Phenomena*, 47(1):263–272, 1991.

37 David H. Wolpert. The stochastic thermodynamics of computation. *Journal of Physics A: Mathematical and Theoretical*, 52(19):193001, 2019.

38 Hui Xie, Mengmeng Sun, Xinjian Fan, Zhihua Lin, Weinan Chen, Lei Wang, Lixin Dong, and Qiang He. Reconfigurable magnetic microrobot swarm: Multimode transformation, locomotion, and manipulation. *Science Robotics*, 4(28):eaav8006, 2019.

# When Should You Wait Before Updating?

## Toward a Robustness Refinement

**Swan Dubois**
Sorbonne Université, CNRS, LIP6, DELYS, Paris, France

**Laurent Feuilloley**
Univ Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, Villeurbanne, France

**Franck Petit**
Sorbonne Université, CNRS, LIP6, DELYS, Paris, France

**Mikaël Rabie**
Université Paris Cité, CNRS, IRIF, Paris, France

**Abstract**

Consider a dynamic network and a given distributed problem. At any point in time, there might exist several solutions that are equally good with respect to the problem specification, but that are different from an algorithmic perspective, because some could be easier to update than others when the network changes. In other words, one would prefer to have a solution that is more robust to topological changes in the network; and in this direction the best scenario would be that the solution remains correct despite the dynamic of the network.

In [6], the authors introduced a very strong robustness criterion: they required that for any removal of edges that maintain the network connected, the solution remains valid. They focus on the maximal independent set problem, and their approach consists in characterizing the graphs in which there exists a robust solution (the existential problem), or even stronger, where any solution is robust (the universal problem). As the robustness criteria is very demanding, few graphs have a robust solution, and even fewer are such that all of their solutions are robust. In this paper, we ask the following question: *Can we have robustness for a larger class of networks, if we bound the number $k$ of edge removals allowed*?

To answer this question, we consider three classic problems: maximal independent set, minimal dominating set and maximal matching. For the universal problem, the answers for the three cases are surprisingly different. For minimal dominating set, the class does not depend on the number of edges removed. For maximal matching, removing only one edge defines a robust class related to perfect matchings, but for all other bounds $k$, the class is the same as for an arbitrary number of edge removals. Finally, for maximal independent set, there is a strict hierarchy of classes: the class for the bound $k$ is strictly larger than the class for bound $k + 1$.

For the robustness notion of [6], no characterization of the class for the existential problem is known, only a polynomial-time recognition algorithm. We show that the situation is even worse for bounded $k$: even for $k = 1$, it is NP-hard to decide whether a graph has a robust maximal independent set.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics

**Keywords and phrases** Robustness, dynamic network, temporal graphs, edge removal, connectivity, footprint, packing/covering problems, maximal independent set, maximal matching, minimum dominating set, perfect matching, NP-hardness

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.7

**Funding** ANR SKYDATA (ANR-22-CE25-0008-02) ANR GrR (ANR-18-CE40-0032).

**Acknowledgements** We thank Nicolas El Maalouly for fruitful discussions about perfect matchings and Dennis Olivetti for pointing out reference [10].

© Swan Dubois, Laurent Feuilloley, Franck Petit, and Mikaël Rabie;
licensed under Creative Commons License CC-BY 4.0



## 1 Introduction

In the field of computer networks, the phrase "*dynamic networks*" refers to many different realities, ranging from static wired networks in which links can be unstable, up to wireless ad hoc networks in which entities directly communicate with each other by radio. In the latter case, entities may join, leave, or even move inside the network at any time in completely unpredictable ways. A common feature of all these networks is that communication links keep changing over time. Because of this aspect, algorithmic engineering is far more difficult than in fixed static networks. Indeed, solutions must be able to adapt to incessant topological changes. This becomes particularly challenging when it comes to maintaining a single leader [3] or a (supposed to be) "static" covering data structure, for instance, a spanning tree, a node coloring, a Maximal Independant Set (MIS), a Minimal Dominating Set (MDS), or a Maximal Matching (MM). Most of the time, to overcome such topological changes, algorithms compute and recompute their solution to try to be as close as possible to a correct solution in all circumstances.

Of course, when the network dynamics is high, meaning that topological changes are extremely frequent, it sometimes becomes impossible to obtain an acceptable solution. In practice, the correctness requirements of the algorithm are most often relaxed in order to approach the desired behavior, while amortizing the recomputation cost. Actually, this sometimes leads to reconsider the very nature of the problems, for example: looking for a "moving leader", a leader or a spanning tree per connected component, a temporal dominated set, an evolving MIS, a best-effort broadcast, *etc.* – we refer to [3, 4] for more examples.

In this paper, we address the problem of network dynamics under an approach similar to the one introduced in [1, 6]: *To what extent of network dynamics can a computation be performed without relaxing its specification?* Before going any further into our motivation, let us review related work on which our study relies.

Numerous models for dynamic networks have been proposed during the last decades–refer to [3] for a comprehensive list of models– some of them aiming at unifying previous modeling approaches, mainly [4, 11]. As is often the case, in this work, the network is modeled as a graph, where the set of vertices (also called nodes) is fixed, while the communication links are represented by a set of edges appearing and disappearing unexpectedly over the time. Without extra assumptions, this modeling includes all possibilities that can occur over the time, for example, the network topology may include no edges at some instant, or it may also happen that some edge present at some time disappears definitively after that. According to different assumptions on the appearance and disappearance (frequency, synchrony, duration, etc.), the dynamics of temporal networks can be classified in many classes [4].

One of these classes, Class $\mathcal{TC}^{\mathcal{R}}$, is particularly important. In this class, a temporal path between any two vertices appears infinitely often. This class is arguably the most natural and versatile generalization of the notion of connectivity from static networks to dynamic networks: every vertex is able to send (not necessarily directly) a message to any other vertex at any time.

For a dynamic network of the class $\mathcal{TC}^{\mathcal{R}}$ on a vertex set $V$, one can partition $V \times V$ into three sets: the edges that are present infinitely often over the time –called *recurrent* edges–, the edges that are present only a finite number of times –called *eventually absent* edges–, and the edges that are never present. The union of the first two sets defines a graph called the *footprint* of the network [4], while its restriction to the edges that are infinitely often present is called the *eventual footprint* [2]. In [2], the authors prove that Class $\mathcal{TC}^{\mathcal{R}}$ is actually the set of dynamic networks whose *eventual footprint* is connected.

In conclusion, from a distributed computing point of view, it is more than reasonable to consider only dynamic networks such that some of their edges are recurrent and their union does form a *connected* spanning subgraph of their footprint.

Unfortunately, it is impossible for a node to distinguish between a recurrent and an eventually absent edge. Therefore, the best the nodes can do is to compute a solution relative to the footprint, hoping that this solution still makes sense in the eventual footprint, whatever it is. In [6], the authors introduce the concept of *robustness* to capture this intuition, defined as follows:

▶ **Definition 1** (Robustness). *A property $P$ is robust over a graph $G$ if and only if $P$ is satisfied in every connected spanning subgraph of $G$ (including $G$ itself).*

Another way to phrase this definition is to say that *a property $P$ is robust if it is still satisfied when we remove any number of edges, as long as the graph stays connected.*

In [6], the authors focus on the problem of maximal independent set (MIS). That is, they study the cases where a set of vertices can keep being an MIS even if we remove edges. They structure their results around two questions:

**Universal question:** For which networks are *all the solutions* robust against any edge removals that do not disconnect the graph?

**Existential question:** For which networks does *there exist a solution* that is robust against any edge removals that do not disconnect the graph?

The authors in [6] establish a characterization of the networks that answer the first questions for the MIS problem. Still for the same problem, they provide a polynomial-time algorithm to decide whether a network answers the second question.

Note that the study of robustness was also very recently addressed for the case of metric properties in [5]. In that paper, the authors show that deciding whether the distance between two given vertices is robust can be done in linear time. However, they also show that deciding whether the diameter is robust or not is coNP-complete.

## 1.1 Our approach

Our goal is to go beyond [6], and to get both a more fine-grained and a broader understanding of the notion of robustness.

Let us start with the fine-grain dimension. In [6], a solution had to be robust against any number of edge removals as long as the graph remains connected. In this paper, we want to understand what are the structures that are robust against $k$ edge removals while keeping the connectivity constraint, for any specific $k$, adding granularity to the notion. We call this concept $k$-robustness (see formal definition below) and we focus on the universal and the existential question of [6] for this fined-grain version of the robustness.

Now for the broader dimension, let us discuss the problems studied. In [6], the problem studied is MIS, which is a good choice in the sense that it leads to an interesting landscape. Indeed, robustness being a very demanding property, one has to find problems to which it can apply without leading to trivial answers. In this direction, one wants to look at local problems, because a modification will only have consequences in some neighborhood and not on the whole graph, which leaves the hope that it actually does not affect the correctness at all. Among the classic local problems, as studied in the LOCAL model (see [9] for the original definition and [8] for a recent book), there are mainly coloring problems and packing/covering problems. The coloring problems (with a fixed number of colors) are not meaningful in

our context: an edge removal can only help. But the packing/covering problems are all interesting, thus we widen the scope to cover three classic problems in this paper: maximal independent set (MIS) as before, but also maximal matching (MM) and minimal dominating set (MDS).

To help the reader grasp some intuition on our approach, let us illustrate the 1-robustness for the maximal matching, *i.e.* a set of edges that do not share vertices and that is maximal in the sense that no edge can be added. To be 1-robust, a matching must still be maximal after the removal of *one arbitrary* edge that does not disconnect the graph. Let us go over various configurations illustrated in Figure 1 (the matched edges are bold ones).

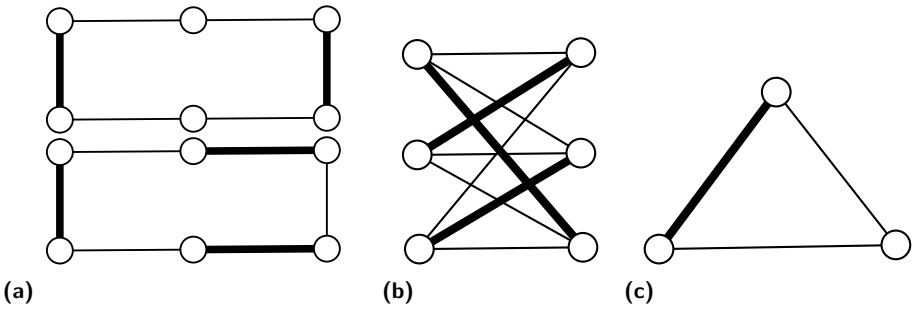



**Figure 1** Three examples of MMs in various graphs.

For the two graphs in Figure 1a, that are cycles of 6 vertices, we can observe that two instances of maximal matching can have different behaviors. Indeed, in the top one, if we remove one matched edge, we are left with a matching that is not maximal in the new graph: the two edges adjacent to the removed one could be added. By contrast, in the bottom graph, any edge removal leaves a graph that is still a maximal matching. Now, in the graph of Figure 1b, a complete balanced bipartite graph, all the maximal matchings are identical up to isomorphism. After one arbitrary edge removal, we are left with a graph where no new edge can be matched. Therefore in this graph, any matching is robust to one edge removal. Note that this is not true for any number of edge removals, illustrating the fact that $k$-robustness and robustness are not equivalent. Finally, in Figure 1c, all the maximal matchings consists of only one edge, and they are not robust to an edge removal. Indeed, after the matched edge is removed, one can choose any of the two remaining ones.

To summarize, Figure 1 illustrates the effect of 1-robustness in three different cases: one where *some* matchings are 1-robust, one where *all* matchings are 1-robust, and one where *no* matching is 1-robust.

## 1.2 Our results

Our first contribution is to introduce the fine-grained version of robustness in Section 2. After that, every technical section of this paper is devoted to provide some answer to the fine-grained version of one of the two questions highlighted above (existential *vs.* universal) for one of the problems we study. Our focus is actually in understanding how do the different settings compare, in terms of both problems and number of removable edges.

Let us start with the universal question. Here, we prove that the three problems have three different behaviors.

For minimal dominating set, the class of the graphs for which any solution is $k$-robust is exactly the same for every $k$ (a class that already appeared in [6] under the name of *sputnik graphs*) as proved in Section 3.

For maximal matching, the case of $k = 1$, which we used previously as an example, is special and draws an interesting connection with perfect matchings, but then the class is identical for every $k \geq 2$. These results are presented in Section 4.

Finally, for maximal independent set, we show in Section 5 that there is a strict hierarchy: the class for $k$ edge removals is strictly smaller than the one for $k - 1$. For this case, we do not pinpoint the exact characterization, but give some additional structural results on the classes.

The existential question is much more challenging. Section 6 presents some preliminary results on the study of this question. For maximal independent set, we show that for any $k$, deciding whether a graph has a maximal independent set that is robust to $k$ edge removals is NP-hard. This is the first NP-hardness result for this type of question.

## 2 Model, definitions, and basic properties

In the paper, except when stated otherwise, the graph is named $G$, the vertex set $V$ and the edge set $E$.

### 2.1 Robustness and graph problems

The key notion of this paper is the one of $k$-robustness.

▶ **Definition 2.** *Given a graph problem and a graph, a solution is* $k$-robust *if after the removal of* at most $k$ *edges, either the graph is disconnected, or the solution is still valid.*

Note that $k$-robustness is about removing at most $k$ edges, not exactly $k$ edges.

We will abuse notation and write $\infty$-robust when mentioning the notion of robustness from [6], with an unbounded number of removals. Hence $k$ is a parameter in $\mathbb{N} \cup \infty$.

▶ **Notation 3.** *We define* $\mathcal{U}_P^k$ *and* $\mathcal{E}_P^k$ *the following way:*

- *Let* $\mathcal{U}_P^k$ *be the class of graphs such that any solution to the problem* $P$ *is* $k$*-robust.*
- *Let* $\mathcal{E}_P^k$ *be the class of graphs such that there exists a solution to the problem* $P$ *that is* $k$*-robust*

Note that to easily incorporate the parameter $k$, we decided to not follow the exact same notations as in [6].

#### Graph problems

We consider three graph problems:

1. Minimal dominating set (MDS): Select a minimal set of vertices such that every vertex of the graph is either in the set or has a neighbor in the set.
2. Maximal matching (MM): Select a maximal set of edges such that no two selected edges share endpoint.
3. Maximal independent set (MIS): Select a maximal set of vertices such that no two selected vertices share an edge.

A *perfect matching* is a matching where every vertex is matched. We will also use the notion of *$k$-dominating set*, which is a set of selected vertices such that every vertex is either selected or is adjacent to two selected vertices. Note that $k$-dominating set sometimes refer to another notion related to the distance to the selected vertices, but this is not our definition.

### The case of robust maximal matching

For maximal matching, the definition of robustness may vary. The definition we take is the following. A maximal matching $M$ of a graph $G$ is $k$-robust if after removing any set of at most $k$ edges such that the graph $G$ is still connected, what remains of $M$ is a maximal matching of what remains of $G$.

## 2.2 Graph notions

We list a few graph theory definitions that we will need.

▶ **Definition 4.** *The* neighborhood *of a node $v$, denoted $N(v)$, is the set of nodes that are adjacent to $v$. The* closed neighborhood *of a node $v$, denoted $N[v]$, is the neighborhood of $v$, plus $v$ itself.*

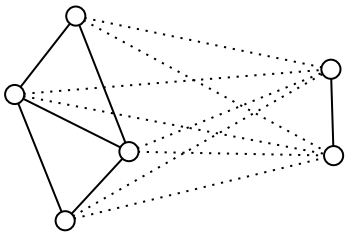
▶ **Definition 5.** *A graph is $t$-(edge)-connected if, after the removal of any set of $(t-1)$ edges, the graph is still connected. A $t$-(edge)-connected component is a subgraph that is $t$-(edge)-connected.*

In the following we are only interested in edge connectivity thus we will simply write *$t$-connectivity* to refer to *$t$-edge-connectivity*. In our proofs, we will use the following easy observation multiple times : in a 2-connected graph every vertex belongs to a cycle.

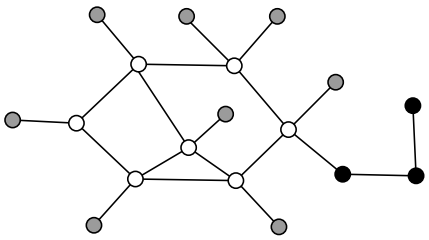
▶ **Definition 6.** *In a connected graph, a* bridge *is an edge whose removal disconnects the graph.*

▶ **Definition 7.** *Given two graphs $G$ and $H$, the* join *of these two graphs, $join(G, H)$, is the graph made by taking the union of $G$ and $H$, and adding all the possible edges $(u, v)$, with $u \in G$ and $v \in H$. See Figure 2a.*

▶ **Definition 8.** *A* sputnik graph *([6]) is a graph where every node that is part of a cycle has an* antenna*, that is a neighbor with degree 1. See Figure 2b.*



**(a)** The join of two graphs: the black edges are the original edges, the doted edges are the one added by the join operation.



**(b)** A sputnik graph. The white vertices are part a cycles, the grey vertices are their antennas, and the black vertices do not belong to any cycle, nor are antennas.

**Figure 2** Illustration of the definitions of Subsection 2.2.

## 2.3 Basic properties

The following properties follow from the definitions.

▶ **Property 9.** *For any problem $P$, for any $k$, $\mathcal{U}_P^{k+1} \subseteq \mathcal{U}_P^k$ and $\mathcal{E}_P^{k+1} \subseteq \mathcal{E}_P^k$.*

In particular, $\mathcal{U}_P^\infty \subseteq \mathcal{U}_P^k \subseteq \mathcal{U}_P^1$ and $\mathcal{E}_P^\infty \subseteq \mathcal{E}_P^k \subseteq \mathcal{E}_P^1$, for all $k$.

▶ **Property 10.** *If a graph is $(k+1)$-*connected *then a solution is $k$-robust if and only if after the removal of any set of $k$ edges the solution is still correct.*

## 3 Minimal dominating set

▶ **Theorem 11.** *For all $k$ in $\mathbb{N} \cup \infty$, $\mathcal{U}^k_{MDS}$ is the set of sputnik graphs.*

**Proof.** We know from [6] that the theorem holds for $k = \infty$. Hence, thanks to Property 9, it is sufficient to prove that the theorem is true for $k = 1$. For the sake of contradiction, consider a graph $G$ in $\mathcal{U}^1_{MDS}$ that is not a sputnik graph. Then there is a node $u$ that belongs to a cycle, and that has no antenna. Let $S$ be the closed neighborhood of $u$, $S = N[u]$. We say that a node of $S$, different from $u$, is an *inside node* if it is only connected to nodes in $S$. We now consider two cases depending on whether there is an inside node or not. See Figure 3.

1. Suppose there exists an inside node $v$. Note that $v$ has at least one neighbor different from $u$ because otherwise it would be an antenna. Let the set $W$ be the closed neighborhood of $v$, except $u$. The set $D = V \setminus W$ is a dominating set of the graph, because all the nodes either belong to $D$ or are neighbors of $u$ (which belongs to $D$). Now, we transform $D$ into a *minimal* dominating set greedily: we remove nodes from $D$ in an arbitrary order, until no more nodes can be removed without making $D$ non-dominating. We claim that this minimal dominating set is not 1-robust. Indeed, if we remove the edge $(u, v)$, $v$ is not covered any more (none of its current neighbors belongs to $D$), and the graph is still connected (because $v$ has a neighbor different from $u$).
2. Suppose there is no inside vertex. Let $a$ be a neighbor of $u$ in the cycle. Let $W$ be the set $S \setminus a$. Again we claim that $V \setminus W$ is a dominating set. Indeed, because there is no inside node, every node in $S$ different from $u$ is covered by node outside $W$, and $u$ is covered by $a$, which belongs to $V \setminus W$. As before we can make this set an MDS by removing nodes greedily, and again we claim it is not 1-robust. Indeed, if we remove the edge $(u, a)$, we do not disconnect the graph (because of the cycle containing $u$), and $u$ is left uncovered. ◀

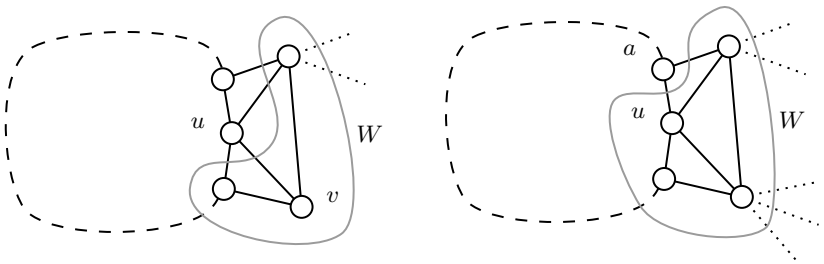



**Figure 3** The two cases of the proof of Theorem 11: with an inside node, on the left, and without an inside node on the right. The cycle is represented by the dashed line, and the dotted lines represent outgoing edges of non-inside nodes.

## 4 Maximal matching

We now turn our attention to the problem of maximal matching, and get the following theorem.

▶ **Theorem 12.** *The class $\mathcal{U}^1_{MM}$ is composed of the set of trees, of balanced complete bipartite graphs, and of cliques with an even number of nodes. For any $k \geq 2$, the class $\mathcal{U}^k_{MM}$ is composed of the cycle on four nodes and of the set of trees.*

The core of this part is the study of the case where only one edge is removed. At the end of the section we consider the more general technically less interesting case of multiple edges removal.

## 4.1 One edge removal

In this subsection we characterize the class of graphs where every maximal matching is 1-robust.

▶ **Lemma 13.** *$\mathcal{U}^1_{MM}$ is composed of the set of trees, of balanced complete bipartite graphs, and of cliques with an even number of nodes.*

The rest of this subsection is devoted to the proof of Lemma 13.

### A result about perfect matchings

The core of the proof is to show a connection to perfect matchings. Once this is done, we can use the following theorem from [10].

▶ **Theorem 14** ([10])**.** *The class of graphs such that any maximal matching is perfect is the union of the balanced complete bipartite graphs and of the cliques of even size.*

### First inclusion

We start with the easy direction of the theorem, which is to prove that the graphs we mentioned are in $\mathcal{U}^1_{MM}$. In trees, any property is robust, since no edge can be removed without disconnecting the graph. For the two other types, we will use the following claim.

▷ Claim 15. Perfect matchings are 1-robust maximal matchings.

Proof. Consider a perfect matching in a graph, and remove an arbitrary edge (that does not disconnect the graph). If this edge was not in the matching, and then we still have a perfect matching, thus a maximal matching. If this edge was in the matching, then there are only two non-matched nodes in the graph (the ones that were adjacent to the edge), and all their neighbours are matched, thus the matching is still maximal. This proves the claim. ◁

In balanced complete bipartite graphs and cliques of even size, any maximal matching is perfect (Theorem 14), and since perfect matchings are 1-robust maximal matchings, we get the first direction of Lemma 13.

### Second inclusion: three useful claims

We now tackle the other direction. The following lemma establishes a local condition that 1-robust matchings must satisfy. See Figure 4 for an illustration.

▷ Claim 16. In a 1-robust maximal matching $M$, if a node $u$ is not matched, then all the nodes of $N(u)$ are matched, and their matched edges are bridges of the graph.

Proof. The fact that all the nodes in $N(u)$ are matched follows from $M$ being a maximal matching. Now, suppose that there exists $(v, w) \in M$, such that $v \in N(u)$ and $(v, w)$ is not a bridge. In other words, the removal of $(v, w)$ does not disconnect the graph. After this removal, both $u$ and $v$ are unmatched, and since $(u, v)$ is an edge of the graph, the matching in the new graph cannot be maximal. This contradicts the 1-robustness of $M$, and proves the lemma. ◁

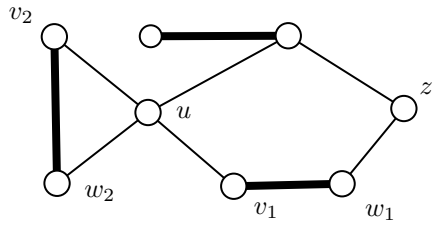



**Figure 4** Illustration of Claim 16. Here we have a maximal matching, and in particular all the neighbors of $u$ are matched, but it is not a 1-robust matching. Indeed, removing $(v_1, w_1)$ gives the possibility of adding $(u, v_1)$ and $(w_1, z)$. Also, having a triangle with a matched edge and an unmatched node, like $(u, v2, w_2)$ is impossible (Claim 17), since removing $(v_2, w_2)$ gives the possibility of adding either $(u, v_2)$ or $(u, w_2)$ to the matching, contradicting the maximality. Hence we need the bridge condition.

The following claim follows directly from Claim 16.

▷ Claim 17. In a 1-robust maximal matching $M$, if there is an unmatched node $u$, two nodes $a, b \in N(u)$ with $(a, b) \in E$, then $(a, b) \notin M$.

We now study the shape of 1-robust maximal matchings in cycles.

▷ Claim 18. In every maximal matching of a graph in $\mathcal{U}^1_{MM}$, if a node belongs to a cycle, then it is matched.

Proof. Our proof of Claim 18 consists in proving that if a maximal matching does not satisfy the condition, then either it is not 1-robust, or we can use it to build another maximal matching that is not 1-robust. In both cases this means the graph was not in $\mathcal{U}^1_{MM}$.

Consider a node $u$ in a cycle. Let $a$ and $b$ be its direct neighbors in the cycle, and let its other neighbors be $(c_i)_i$. There can be several configurations, with $a$ adjacent to $b$ or not, etc. The proof is generic to all these cases, but Figure 5 illustrates different cases. Consider a 1-robust maximal matching $M$ where $u$ is unmatched. Because of Claim 16, we know that there exists nodes $a'$, $b'$, and $c'_i$ for all $i$, such that respectively $(a, a')$, $(b, b')$ and $(c_i, c'_i)$ (for all $i$) are bridges of the graph. Because of the bridge condition, these nodes $a'$, $b'$ and $c'_i$ (for all $i$) are all different, and are different from $a$, $b$, $u$ and the $c_i$'s. Let us also denote $d$ the neighbor of $a$ in the cycle that is not $u$. Note that $d$ can be a $c_i$ or $b$, but no other named node. (See Figure 5 for an illustration.) Now we create a new matching $M'$ from $M$ in the following way. First remove all the edge of the matching that are not adjacent to one of the nodes above. Then, remove $(a, a')$ and any edge matching $d$ (if it exists). Note that this last edge matching $d$ could be a $(c_j, c'_j)$ or $(b, b')$. Add $(a, d)$ to the matching (note that both nodes are unmatched before this operation). In this matching, all the neighbors of $u$ are matched. We complete this matching into a maximal matching $M'$. The edge $(a, d)$ is in $M'$ and $u$ is unmatched, which is a contradiction with Claim 16, thus $M'$ cannot be 1-robust, and this proves the claim. ◁

## Second inclusion: putting pieces together

▷ Claim 19. A graph in the class $\mathcal{U}^1_{MM}$ is either a tree or is 2-connected.

Proof. Consider a graph that is neither a tree nor a 2-connected graph. There necessarily exists a bridge $(u, v)$ such that $u$ belongs to a cycle. We distinguish two cases.

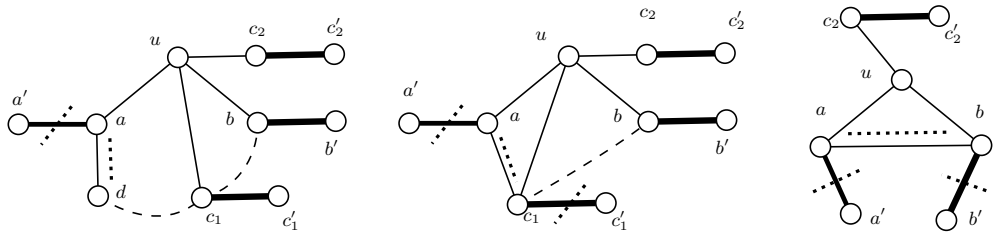



**Figure 5** Illustration of the proof of Claim 19, in three cases: $d$ is not $b$ nor a $c_i$, $d$ is one of the $c_i$, $d$ is $b$. The dashed lines represent paths with at least one edge. The dotted lines represent the change we operate: the edges that are crossed out are removed from the matching, the edges that are have a dotted double are added to the matching.

1. Node $v$ is linked only to $u$, that is, $v$ is a pendant node. Then we build a maximal matching $M$ by first forcing $u$ to be matched to a node that is not $v$, and then completing it greedily. Now, if we remove the edge that matches $u$, we do not disconnect the graph since $u$ was part of a cycle, but neither $u$ nor $v$ is matched, thus the matching is not maximal ($(u, v)$ could be added). Thus the matching $M$ was not 1-robustness.
2. Node $v$ is linked to another node $w$. Let consider $(v_i)_i$ the set of nodes such that $v_i \neq v$ and $(u, v_i)$ is a bridge. By the previous point, we know that there exists some $w_i \neq u$ in $N(v_i)$. Moreover, those $(w_i)$ must be distinct pairwise and from all the other named nodes, otherwise $(u, v_i)$ would not have been a bridge. The node $w$ and the nodes $(w_i)_i$ cannot be part of the 2-connected component of $u$, otherwise $(u, v)$ and $(u, v_i)$ would not be a bridge. We build a maximal matching $M$ by first forcing $(u, v)$ and $(v_i, w_i)$ for all i, and then completing it greedily. As observed earlier, in the 2-connected component of $u$ every node must belong to a cycle, thus by Claim 18, we get that every node of this component must be matched. We now build a second matching $M'$. We start from $M$ and remove from the matching $(u, v)$ and every edge that is in $v$'s side of the bridge. Then we force $(v, w)$ in the matching, and complete it greedily. The matching $M'$ is maximal and $u$ is unmatched, since all of its neighbors are matched, hence by Claim 18 it is not 1-robust, since it belongs to a 2-connected components thus to a cycle.

This concludes the proof of the claim. ◁

To conclude a graph in the class is either a tree, or is 2-connected, and in this last case because of Claim 18, every node must be matched in every maximal matching. Then Lemma 13 follows from Theorem 14.

## 4.2 More than one edge removal

▶ **Lemma 20.** *For any $k \geq 2$, $\mathcal{U}_{MM}^k$ is composed of the cycle on four nodes and of the set of trees.*

**Proof.** We first prove the reverse inclusion. As before, trees are in $\mathcal{U}_{MM}^k$ for any $k$ because any edge removal disconnects the graph. Then for $C_4$, note that it belongs to $\mathcal{U}_{MM}^1$, and that the removal of more than one edge disconnects the graph.

For the other direction, we can restrict to $\mathcal{U}_{MM}^2$, and by definition it is included in $\mathcal{U}_{MM}^1$. Thus we can simply study the case of the balanced complete bipartite graphs and of the cliques on an even number of nodes. Consider first a complete bipartite graph $B_{k,k}$ with $k > 2$ (that is any $B_{k,k}$ larger than $C_4$), and a maximal matching $M$. Take two arbitrary edges $(a_1, b_1)$ and $(a_2, b_2)$ from the matching and remove them from the graph. The graph

is still connected. Now the nodes $a_1$ and $b_2$ are unmatched and there is an edge between them, thus the resulting matching is not maximal and $M$ is not 2-robust. Thus the only $B_{k,k}$ left in the class $\mathcal{U}^2_{MM}$ is $C_4$. For the cliques on an even number of nodes, consider one that has strictly more than two vertices. A maximal matching $M$ contains at least two edges $(u_1, v_1)$ and $(u_2, v_2)$. When we remove these edges from the graph, we still have a connected graph, $u_1$ and $u_2$ are unmatched, but $(u_1, u_2)$ still exists, thus the resulting matching is not maximal and $M$ was not 2-robust. ◀

# 5 Maximal independent set

Maximal independent set illustrates yet another behavior for the classes $(\mathcal{U}^k_{MIS})_k$: they form an infinite strict hierarchy.

## 5.1 An infinite hierarchy

▶ **Theorem 21.** *For every $k \geq 1$, $\mathcal{U}^{k+1}_{MIS}$ is strictly included in $\mathcal{U}^k_{MIS}$.*

**Proof.** Let $k \geq 1$. We will define a graph $G_k$, and prove that it belongs to $\mathcal{U}^k_{MIS}$ but not to $\mathcal{U}^{k+1}_{MIS}$.

To build $G_k$, consider a bipartite graph with $k+2$ nodes on each of the sides $A$ and $B$, and add a pendant neighbor $v$ to a node $u$ on the side $A$. See Figure 6. This graph has only three MIS: $A$, $v \cup B$, and $v \cup (A \setminus u)$. Indeed: (1) if the MIS contains $u$, then it cannot contain vertices outside of $A$, and to be maximal it contains all of $A$, (2) if it contains a vertex of $B$, it cannot contain a vertex of $A$, and by maximality it contains all of $B$ and $v$, and (3) if it contains $v$, and no vertex of $B$, then by maximality it is $v \cup (A \setminus u)$.

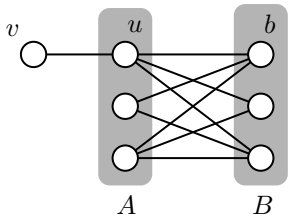



**Figure 6** . Illustration of the graph $G_k$ in the proof of Theorem 21.

We claim that these three MIS are $k$-robust, therefore $G_k$ is in $\mathcal{U}^k_{MIS}$. Suppose an MIS is not $k$-robust. Then there exists a vertex $w$ that is not part of the MIS, such that after at most $k$ edge removals, it has no neighbor in the MIS anymore. Let us make a quick case analysis depending on who is this vertex $w$. It cannot be $v$, since removing the edge $(u, v)$ would disconnect the graph. It cannot be a vertex of $A$, nor of $B$, because in all MIS mentioned, all non selected nodes (except $v$) have at least $k+1$ selected neighbors.

Now we claim that $v \cup (A \setminus u)$ is not $(k+1)$-robust, thus $G_k$ does not belong to $\mathcal{U}^{k+1}_{MIS}$. We choose a vertex $b$ on the $B$ side, and remove all the edges $(a, b)$ for $a \in A \setminus u$. This is a set of $k+1$ edges whose removal does not disconnect the graph, but leaves $b$ without selected neighbors. This $v \cup (A \setminus u)$ is not $(k+1)$-robust. ◀

## 5.2 A structure theorem for $\mathcal{U}^k_{MIS}$

The construction used in the proof of Theorem 21 is very specific and does not really inform about the nature of the graphs in $\mathcal{U}^k_{MIS}$. It can be generalized, with antennas on both sides and arbitrarily large (unbalanced) bipartite graphs with arbitrary number of antennas per

nodes, but it is still specific. Moreover these construction heavily rely on pendant nodes, that are in some sense abusing the fact that we do not worry about the correctness of the solution if the graph gets disconnected.

In order to better understand these classes, and to give a more flexible way to build such graphs, we prove a theorem about how the class behaves with respect to the join operation (Definition 7).

We denote by $\mathcal{G}_p$ the class of graphs where every maximal independent set has size *at least* $p$. We say that a graph class is *stable by an operation* if, by applying this operation to any (set of) graph(s) from the class, the resulting graph is also in the class.

▶ **Theorem 22.** *For all $k$, the class $\mathcal{U}^k_{MIS} \cap \mathcal{G}_{k+1}$ is stable by join operation. Also, if either $G$ or $H$ is not in $\mathcal{U}^{k+1}_{MIS}$, then $join(G, H)$ is not in $\mathcal{U}^{k+1}_{MIS}$ either.*

**Proof.** Let us start with the first statement of the theorem. Consider two graphs $G$ and $H$ in $\mathcal{U}^k_{MIS} \cap \mathcal{G}_{k+1}$. We prove that $J = join(G, H)$ is also in $\mathcal{U}^k_{MIS} \cap \mathcal{G}_{k+1}$.

▷ Claim 23. Any MIS of $J$ is either completely contained in the vertex set of $G$, and is an MIS of $G$, or contained in the vertex set of $H$, and is an MIS of $H$.

Proof. Consider an independent set in $J$. If it has a node $u$ in $G$, then it has no node in $H$, as by construction, all nodes of $H$ are linked to $u$. The analogue holds if the independent set has a node in $H$. Thus any independent set is either completely contained in $G$ or completely contained in $H$. Now, a set is maximal independent in $G$ (resp. $H$) alone if and only if it is maximal independent in $G$ (resp. $H$) inside $J$. Indeed the only edges that we have added are between nodes of $G$ and nodes of $H$. This proves the claim. ◁

Therefore, the resulting graph is in $\mathcal{G}_{k+1}$. Now for the $k$-robustness, consider without loss of generality an MIS of $J$ that is in part $G$, and suppose it is not $k$-robust. In this case there must exists a non-selected vertex $v$, that has no more selected neighbors after the removal of $k$ edges (while the graph stays connected). This node cannot be in the part $G$, otherwise the same independent set in the graph $G$ would not be $k$-robust. And it cannot be in the part $H$, since every node of $H$ is linked to all the vertices of the MIS, and this set has size at least $k + 1$ since $G \in \mathcal{G}_{k+1}$.

Now, let us move on to the second statement of the theorem. Let's assume that $G$ has an MIS $S$ and $k + 1$ edges such that their removal makes that $S$ is not longer maximal (i.e. there exists some $u$ that can be added to the set). Then, $S$ is also an MIS of $join(G, H)$, and the removal of the same edges will allow to add $u$ to the set, as the only new neighbors of $u$ are from $H$ that does not contain any node of the chosen MIS ◀

## 6 The existence of a robust MIS is NP-hard

Remember that we have defined two types of graph classes related to robustness. For a given problem, and a parameter $k$, the universal class is the class where every solution is $k$-robust. This is the version we have explored so far. For this version, recognizing the graphs of the class is easy since these have simple explicit characterization. The second type of class is the existential type, where we want that there exists a solution that is $k$-robust. And here the landscape is much more complex. Indeed, in [6] in the simpler case of robustness without parameter, there is no explicit characterization of the existential class, only a rather involved algorithm. In this section we show that, when we add the parameter $k$ the situation becomes even more challenging: the algorithm of [6] runs in polynomial time, and here we show that the recognition of $\mathcal{E}^1_{MIS}$ is NP-hard.

▶ **Theorem 24.** *For every odd integer $k$, it is NP-hard to decide whether a graph belongs to $\mathcal{E}^k_{MIS}$.*

The rest of this section is devoted to the proof of this theorem. It is based on the NP-completeness of the following problem.

Perfect stable
Input: A graph $G = (V, E)$.
Question: Does there exists a subset of vertices $S \subset V$ that is independent 2-dominating?

Remember that a set is independent 2-dominating if no two neighbors can be selected, and every non-selected vertex should have at least two selected neighbors. Just to get some intuition about why we are interested in this problem, note that with independent 2-dominating after removing an edge between a selected and a non-selected vertex, the non-selected vertex is still dominated. It was proved in [7] that Perfect stable is NP-hard in general. We will need the following strengthening of this hardness result.

▶ **Lemma 25.** *Deciding whether a 2-connected graph has an independent 2-dominating set is NP-complete.*

Note that this lemma does not follow directly from [7] because the reduction there does use some non-2-connected graphs.

**Proof.** Let $G$ be an arbitrary connected graph with at least one edge. Consider $G'$ to be the same as $G$ but with a universal vertex, that is, $G$ with an additional vertex $u$ that is adjacent to all the vertices of $G$. This graph is 2-edge connected. Indeed, since $G$ is connected and has at least two vertices, removing any edge $(u, v)$ with $v \in V(G)$ cannot disconnect the graph, and removing an edge from $G$ does not disconnect the graph because all nodes are linked through $v$.

We claim that $G'$ has an independent 2-dominating set if and only if $G$ has one. First, suppose that $G$ has such a set $S$. Note that the set $S$ has at least two selected vertices. Indeed, $G$ has at least one edge, which implies that at least one vertex is not selected (by independence), and such a vertex should be dominated by at least two selected vertices. Now we claim that $S$ is also a solution for $G'$. Indeed, the addition of $u$ to the graph does not impact the independence of $S$, nor the 2-domination of the nodes of $G$, and $v$ is covered at least twice, since there are at least two selected vertices in $G$. Second, if $G'$ has independent 2-dominating set $S'$, it cannot contain $v$. Indeed, because of the independence condition, if $v$ is selected, then no other node can be selected, and then the 2-domination condition is not satisfied. Then $S'$ is contained in $G$ and is an independent 2-dominating set of $G$. ◀

Now, let us formalise the connection between robustness and independent 2-domination.

▶ **Lemma 26.** *In a 2-connected graph, the 1-robust maximal independent sets are exactly the independent 2-dominating sets.*

**Proof.** As a consequence of Property 10, in a 2-connected graph, a 1-robust MIS is an MIS that is robust against the removal of any edge (that is, we can forget about the preserved connectivity in the robustness definition). This means that every node not in the MIS is covered twice, otherwise one could break the maximality by removing the edge between the node covered only once and the node that covers it. In other words, the independent set must be 2-dominating. For the other direction it suffices to note that any independent dominating set is a maximal independent set. ◀

At that point, plugging Lemma 25 and Lemma 26 we get that deciding whether there exists a 1-robust MIS in a graph is NP-hard, even if we assume 2-connectivity. This last lemma is the final step to prove Theorem 24.

▶ **Lemma 27.** *For any 2-connected graph $G$ and any integer $k > 1$, we can build in polynomial-time a graph $G'$, such that: $G$ has a 1-robust MIS if and only if $G'$ has a $2k-1$-robust MIS.*

**Proof.** We build $G'$ in the following way. Take $k$ copies of $G$, denoted $G_1, ..., G_k$, with the notation that $u_x$ is the copy of vertex $u$ in the x-th copy. For every edge $(u, v)$ of $G$, we add the edge $(u_x, v_y)$ for any pair $x, y \in 1, ..., k$.

Let us first establish the following claim. An MIS in the graph $G'$ necessarily has the following form: it is the union of the exact same set repeated on each copy. Indeed, let $u_i$ be in the MIS. For any $j \neq i$, all the neighbors of $u_j$ in the copy $G_j$ are a neighbor of $u_i$, which implies that they are not in the MIS. Hence, no neighbor of $u$ in any copy can be in the MIS. As those nodes are the only neighbors of $u_j$, it implies that $u_j$ is also in the MIS.

Now suppose that $G$ has a 1-robust MIS. We can select the clones of this MIS in each copy, and build an MIS for $G'$ (the independence and maximality are easy to check). In this MIS of $G'$, every non-selected vertex has at least $2k$ selected neighbors, therefore this MIS is $2k-1$ robust.

Finally, suppose that $G'$ has a $2k-1$ robust MIS. Thanks to the claim above, we know that this MIS is the same set of vertices repeated on each copy. We claim that when restricted to a given copy, this MIS is 1-robust. Indeed, if it were not, then there would be one non-selected vertex with at most one selected neighbor, and this would mean that in $G'$ this vertex would have only $k$ selected neighbors, which contradicts the $2k-1$ robust (given the connectivity). ◀

## 7 Conclusions

In this paper we have developed the theory of robustness in several ways: adding granularity and studying new natural problems to explore its diversity. The next step is to fill in the gaps in our set of results: characterizing exactly the classes $\mathcal{U}^k_{MIS}$, and understanding the complexity of answering the existential question for maximal matching and minimum dominating set. We believe that a polynomial-time algorithm can be designed to answer the existential question in the case of maximal matching with $k = 1$, with an approach similar to the one of [6] for MDS (that is, via a careful dynamic programming work on a tree-like decomposition of the graphs). A more long-term goal is to reuse the insights gathered by studying robustness to help the design of dynamic algorithms.

## References

1 Nicolas Braud Santoni, Swan Dubois, Mohamed Hamza Kaaouachi, and Franck Petit. A generic framework for impossibility results in time-varying graphs. In *IEEE 29nd International Symposium on Parallel and Distributed Processing (IPDPS 2015), 17th Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2015)*, pages 483–489. IEEE, 2015.

2 Nicolas Braud-Santoni, Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. The next 700 impossibility results in time-varying graphs. *International Journal of Networking and Computing*, 6(1):27–41, 2016.

3 A. Casteigts and P. Flocchini. Deterministic algorithms in dynamic networks: Problems, analysis, and algorithmic tools. Technical report, Defence Research and Development Canada, 2013-020, 2013.

4 A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

5 Arnaud Casteigts, Timothée Corsini, Hervé Hocquard, and Arnaud Labourel. Robustness of distances and diameter in a fragile network. In *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022*, volume 221 of *LIPIcs*, pages 9:1–9:16, 2022. doi:10.4230/LIPIcs.SAND.2022.9.

6 Arnaud Casteigts, Swan Dubois, Franck Petit, and John Michael Robson. Robustness: A new form of heredity motivated by dynamic networks. *Theor. Comput. Sci.*, 806:429–445, 2020. doi:10.1016/j.tcs.2019.08.008.

7 Cornelius Croitoru and Emilian Suditu. Perfect stables in graphs. *Inf. Process. Lett.*, 17(1):53–56, 1983. doi:10.1016/0020-0190(83)90091-1.

8 Juho Hirvonen and Jukka Suomela. Distributed algorithms, 2020. Available from: https://jukkasuomela.fi/da2020/da2020.pdf.

9 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

10 David P Summer. Randomly matchable graphs. *Journal of Graph Theory*, 3(2):183–186, 1979. doi:10.1002/jgt.3190030209.

11 B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

# Dynamic Graphs Generators Analysis: An Illustrative Case Study

**Vincent Bridonneau**
Université Le Havre Normandie, Normandie Univ, LITIS EA 4108, 76600 Le Havre, France

**Frédéric Guinand**
Université Le Havre Normandie, Normandie Univ, LITIS EA 4108, 76600 Le Havre, France

**Yoann Pigné**
Université Le Havre Normandie, Normandie Univ, LITIS EA 4108, 76600 Le Havre, France

## Abstract

In this work, we investigate the analysis of generators for dynamic graphs, which are defined as graphs whose topology changes over time. We focus on generated graphs whose orders are neither growing nor constant along time. We introduce a novel concept, called "sustainability," to qualify the long-term evolution of dynamic graphs. A dynamic graph is considered sustainable if its evolution does not result in a static, empty, or periodic graph. To measure the dynamics of the sets of vertices and edges, we propose a metric, named "Nervousness," which is derived from the Jaccard distance. As an illustration of how the analysis can be conducted, we design a parametrized generator, named D3G3 (Degree-Driven Dynamic Geometric Graphs Generator), that generates dynamic graph instances from an initial geometric graph. The evolution of these instances is driven by two rules that operate on the vertices based on their degree. By varying the parameters of the generator, different properties of the dynamic graphs can be produced. Our results show that in order to ascertain the sustainability of the generated dynamic graphs, it is necessary to study both the evolution of the order and the Nervousness for a given set of parameters.

**2012 ACM Subject Classification** Theory of computation → Dynamic graph algorithms; Mathematics of computing → Random graphs; Networks → Topology analysis and generation

**Keywords and phrases** Dynamic Graphs, Graph Generation, Graph Properties, Evolutionary models

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.8

**Funding** Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).

## 1 Introduction

Nature and human societies offer many examples of systems composed of entities that interact, communicate or are just connected with each other. The Internet, a transportation network, a swarm of robots, an ant colony, a social network, a urban network, or a crowd are some examples [2].

Graphs are certainly one of the best formalism for modeling them. Every vertex in the graph models one entity. A link is added between two vertices when a particular condition about the corresponding entities is verified. For instance: two people are talking to each other, a predator catches a prey, two playing cards are in the same hand, a virus passes from one individual to another, two actors perform in the same play, etc. The semantic of the interaction, communication or connection is proper to the system.

During the last two decades, many works have been dedicated to the study of networks modeling these systems. It has been shown that, unlike classical, regular or random graphs, graphs modeling complex real systems present specific statistical properties, leading researchers to introduce the term of complex networks for naming them. Among the main characteristics that were highlighted are the small-world and the scale-free properties. The small-world property was discovered many years ago, in the 60s, when Stanley Milgram


© Vincent Bridonneau, Frédéric Guinand, and Yoann Pigné;
licensed under Creative Commons License CC-BY 4.0

imagined and conducted the "small-world problem" in order to measure, through postal mail, the number of intermediaries between any two persons in the USA [15, 20]. The scale-free property was also observed quite a long time ago for some datasets.

Researchers working on networks are motivated by one key question: "How can we explain the existence of certain properties in these networks?" To answer this, they aim to design generative processes that can produce networks with these properties. In 1998, Watts and Strogatz proposed a rewiring process to generate small-world networks from a regular lattice [21]. In 1999, Barabàsi and Albert introduced the preferential attachment mechanism to create networks with both small-world and scale-free properties [1]. Other works have also used preferential attachment to generate networks with other features, such as soft community structures [9] and navigability [3]. For example, Papadopoulos *et al.* introduced self-similarity [17] as a mechanism that can create soft communities. This new concept aims at connecting entities not because one is popular, but because they share similarities. This work has been possible using hyperbolic geometry which helps embedding both popularity (the preferential attachment) and similarity [19, 13]. Other models using such a geometry have then been investigated creating networks with more properties, such as the Geometric Preferential Attachment [22] and the nonuniform Popularity-similarity Optimization model [16]. One relevant point to highlight is that such processes generate growing networks: at each time step a new vertex is added to the graph and is more likely linked to high degree vertices. Other mechanisms exist where the amount of vertices does not change over time. For instance, this is the case of edge-markovian processes where only edges are changed over time [6]. Other models such as Erdos-Renyi evolution model [7] or the configuration model [4] can also be seen as such a model.

However many real-world systems are composed of a varying number of entities (increasing and decreasing). For instance, living being populations may see the number of individuals increase during some periods and decrease for some other periods [14]. And, to our knowledge, generative models with such a characteristic have not been deeply studied. The work developed in this paper aims at addressing this more general case where the set of vertices, between two consecutive time steps, may either increase or decrease or may change while keeping the same cardinality. To provide an example of the latter case, consider the interaction between players during a game of rugby. The total number of players on the field remains constant at 30 (assuming no exclusions); therefore, the order of the players is fixed. Nevertheless, substitutions occur throughout the game.

Given a generative process, questions asked are: "how the dynamics of generated graphs can be characterized?"; "what metrics might be used for that purpose, and how to compute them?"; "from the point of view of dynamics, is it possible to classify or gather generators into classes or families?". In this ongoing work, not all questions are addressed. But we hope it will be a milestone for carrying out analysis of dynamic graphs generators. For this, in Section 2 a generic model of generators is presented and discussed. It is followed by the definition of a novel notion based on a specific metric, both targeting the dynamics of the graphs. Finally, the Degree-Driven Dynamic Geometric Graph Generator (D3G3) is presented. D3G3 is a parameterized generator and according to the parameters, it can produce a wide variety of dynamics. It will be used as a case study. A first global analysis of the generated graph families is performed in Section 3. Section 4 focuses on specific values of the parameters and present a rigorous analysis of the evolution of the dynamics of the graph and of the likelihood of its sustainability. A conclusion, drawing some perspectives and future investigations, closes temporarily this work.

# 2 Definitions and Generative Model and Definitions

## 2.1 Notations

Consider two sets $A$ and $B$:

- $\triangle$ operator: $A \triangle B$ is defined as $A \cup B - A \cap B$. For instance if $A = \{1, 2, 3, 4, 5\}$ and $B = \{4, 5, 6, 7, 8\}$ then $A \triangle B = \{1, 2, 3, 6, 7, 8\}$
- Consider $G$ a dynamic graph, $G_t = (V_t, E_t)$ denotes the state of the graph at time $t$, where $V_t$ is the set of vertices and $E_t$ is the set of edges.
- $|V_t|$ (resp. $|E_t|$) corresponds to the number of vertices (resp. edges) of graph $G_t$.
- For simplifying notations in the document, $|V_t|$ is often denoted by $n_t$
- $G = (V, E)$ is said to be a null graph if both $V = \emptyset$ and $E = \emptyset$. In the report such a null graph can also be called an empty graph.
- Let $a$ and $b$ to real numbers such that $a < b$. Then $[a, b]$ refers to as a closed interval of real number. Both end points belong to the interval. The open interval $(a, b)$ represents the same object, but end points are not included. A half-open interval $[a, b)$ is an interval including the endpoint $a$ but not $b$. The $(a, b]$ one includes $b$ but not $a$.
- The set $\mathbb{N}$ refers to as the set of non-negative integers ($\{0, 1, \dots\}$). The set of positive integers is referred to as $\mathbb{N}^*$ ($\{1, 2, \dots\}$).

## 2.2 Position of the work with respect to Temporal Networks

Current definitions of temporal networks (TN) include *time-varying graphs* [18], *temporal networks* [12], *evolving graphs* [8], etc. They all define structures described by sequence of static graphs, ordered by a timestamp (e.g., $G = (G_i = (V_i, E_i))_{i \geqslant 0}$) where $i$ refers to the time step). It is worth mentioning that TN definitions do not include information about the generative process. Thus, the way the graph at time $t + 1$ is obtained from the graph at time $t$ is not described. In this report, the emphasis is precisely on the study of generative processes. This work is therefore positioned upstream of TN. In the sequel graphs produced by generators are called dynamic graphs or simply graphs.

## 2.3 Generalities

From a general point of view, a dynamic graph generator can be defined as a process with input data, that produces at each time step $t+1$ a new static graph $G_{t+1}$. It is produced from already generated static graphs $\{G_1, \dots, G_t\}$ and possibly additional information. Thus, the output of a dynamic graph generator is a flow of static graphs identified by time stamps. The time stamps may also corresponds to events, and in such a case, the time interval between two time stamps may be different. However, in this report, for sake of clarity, we consider integer time stamps. If the flow stops, for whatever reason (e.g. clock has been stopped, evolution process is finished) at step $T$, the set of generated static graphs $\{G_1, G_2, \dots, G_T\}$ corresponds to a temporal network (TN).

## 2.4 Sustainability

The goal of this section is to introduce a novel notion for qualifying the dynamics of a graph. Only measurement of the order (or of the density) of the graph is not enough for qualifying its dynamics. For instance, if a dynamic graph becomes static, all vertices remain the same and the graph order does not change. Conversely, if between two consecutive time steps all vertices are replaced by new ones, the order also remains the same, but the dynamics is different. Sustainability qualifies a dynamic graph that never becomes null or periodic (which includes static). A graph owing the sustainability property is said sustainable.

▶ **Definition 1** (Graph sustainability). *A dynamic graph $G$ is said sustainable if both Condition 1 and Condition 2 are not verified.*

*Condition 1:* $\exists T \in \mathbb{N}, \forall t \geqslant T, G_t = (\emptyset, \emptyset)$
*Condition 2:* $\exists T \in \mathbb{N}$ *and* $\exists k \in \mathbb{N}^*, \forall t \geqslant T, G_t = G_{t+k}$

Some well-known graph generators produce sustainable dynamic graphs. For instance generators of growing networks. Indeed, for all $t \in \mathbb{N}$, $|V_t| > |V_{t-1}|$, and $G_t \neq (\emptyset, \emptyset)$. For these generators, graph sustainability is obvious and does not require any analysis.

Unlike these cases, some generators are based on mechanisms making the evolution of the vertices (and edges) more difficult to predict, and the dynamics is worth studying. For that purpose, we propose to consider a metric enabling a quantification of the dynamics.

## 2.5 Nervousness

This metric provides a way of measuring the dynamics of a graph. Note that this metric is derived from the Jaccard distance, which can be defined as one minus the coefficient of community as outlined in [11]. However, in the context of dynamic graphs it seems to us more meaningful to call it nervousness. This metrics is defined at the level of vertices, edges and at the level of the graph. In this work, only vertices nervousness is defined. It is different from the burstiness which is defined at the node/edge level during the lifetime of the graph [10] and aims at representing the frequency of events occurring on each node/edge. Nervousness metric aim is to capture the dynamics of creation and deletion of nodes and edges between two time steps at graph level.

▶ **Definition 2** (Vertices Nervousness). *Given a dynamic graph $G$, such that at time $t$ $G_t = (V_t, E_t)$. We call* **vertices nervousness** *at time $t$ and denoted by $\mathcal{N}^V(t)$, the ratio:*

$$\mathcal{N}^V(t) = \frac{|V_{t+1} \triangle V_t|}{|V_{t+1} \cup V_t|} = \frac{|V_{t+1} \cup V_t - V_t \cap V_{t+1}|}{|V_{t+1} \cup V_t|}$$

This metric is complementary with the graph order measure. Indeed, graph order can remain constant between two consecutive time steps although some vertices change. If all vertices are replaced, nervousness equals 1. If all vertices are kept, nervousness is 0. Similarly we define the edges nervousness as $\mathcal{N}^E(t) = \frac{|E_t \triangle E_{t+1}|}{|E_t \cup E_{t+1}|}$. Accordingly, Graph Nervousness is defined as $\mathcal{N}^G(t) = (\mathcal{N}^V(t), \mathcal{N}^E(t))$.

For illustrating this definition, consider the following cases for a dynamic graph, from $t$ to $t+1$. We denote $|V_t| = n_t$. We also assume that between $t$ and $t+1$ the order remains the same, thus $|V_{t+1}| = n_{t+1} = n_t = n$.

- if all vertices are replaced:

$$\mathcal{N}^V(t) = \frac{|V_t \triangle V_{t+1}|}{|V_t \cup V_{t+1}|} = \frac{2n}{2n} = 1$$

- if half of the vertices are replaced: $\mathcal{N}^V(t) = \frac{3n/2 - n/2}{3n/2} = \frac{n}{3n/2} = \frac{2}{3}$
- if the vertices remain the same, the union of the sets is equal to their intersection thus: $\mathcal{N}^V(t) = 0$

When the order changes, for instance if all vertices are duplicated, thus $|V_{t+1}| = 2n_t = 2n$: $\mathcal{N}^V(t) = \frac{2n-n}{2n} = \frac{1}{2}$

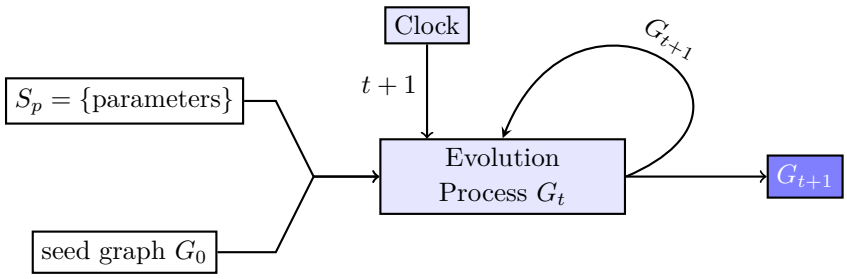
## 2.6 Sustainability vs Nervousness

Sustainability and nervousness are closely related. Sustainability describes a dynamic graph property while nervousness enables the measure of the evolution of vertices and edges sets between two consecutive time steps. When nervousness is null for both sets, the graph is static and thus does not have the sustainability property. Consider a dynamic graph $G$, if for all $t \in \mathbb{N}$, $\mathcal{N}^V(t) \neq 0$ or $\mathcal{N}^E(t) \neq 0$, then $G$ holds the sustainability property, except if $G$ is periodic.

## 2.7 D3G3: definition

In this section we define a parameterized model generating families of dynamic graphs. An instance of the generative model is defined by a set of parameters. For studying the model, we analyze, according to the parameters set, the dynamic graphs families produced and rely on both the sustainability and the nervousness for that purpose.

The generator has two types of inputs: a set of parameters, $S_p$, and an initial graph, called seed graph and denoted $G_0$. At each time step $t+1$ it produces, from the previous graph $G_t$, a new graph $G_{t+1}$ as illustrated on the figure.




Graphs produced by D3G3 are geometric graphs. A geometric graph is defined by an euclidean space and a threshold $d$. For this study, without loss of generality we consider a 2D-unit-torus (i.e., a square $[0;1)^2$ where the two opposite sides are connected). Each vertex is characterized by a set of coordinates, such that given two vertices $u$ and $v$ it is possible to compute their euclidean distance: $dist(u,v)$. Given $V$ the set of vertices, the set of edges $E$ is defined in the following way: $E = \{(u,v) \in V^2 \mid \text{dist}(u,v) \leqslant d\}$. It is important to notice that borders of the square modeling the torus are connected. Therefore considering one node on the torus, the value of $d$ for which the surface of the disk of radius $d$ centered on this node reaches its maximum for $d = \sqrt{2}/2$. It represents the half diagonal of the square.

Graphs generated by D3G3 are produced thanks to an evolution process. This mechanism is parameterized by an initial graph (the seed graph) and by two transition rules driving the evolution of the graph between two consecutive time steps. Apart from a random generator, no external decision or additional information is used by this mechanism. Rules are based on node degrees only and rely on a random generator for positioning new nodes in the 2D euclidean space. This leads to the name of the generator: *Degree-Driven Dynamic Geometric Graphs Generator* or D3G3.

From now, graphs we are studying are referred to as sequences of static graphs $(G_t = (V_t, E_t))_{t \geqslant 0}$, where $t \geqslant 0$ is the time step. The initial graph, $G_0$ $(t = 0)$ is called the seed graph.

▶ **Definition 3** (Degree Driven Dynamic Geometric Graph Generator). *An instance of D3G3 is defined by an initial graph, a set of parameters and two rules:*

- $G_0 \neq (\emptyset, \emptyset)$ *the seed graph,*
- *parameters:*
  - $d \in (0, \frac{\sqrt{2}}{2})$ *(distance threshold for connection),*
  - $S_S$ *a set of non-negative integers*
  - $S_C$ *a set of non-negative integers*
- *rules applied on* $G_t$ *leading to* $G_{t+1}$*:*
  - *if* $v \in V_t$*, then* $v \in V_{t+1}$ *if and only if* $deg(v) \in S_S$ *(conservation rule)*
  - *if* $v \in V_t$ *and if* $deg(v) \in S_C$ *then add a new vertex to* $V_{t+1}$ *with a random position in the unit-torus (creation rule)*

At a given time step, two nodes are connected if and only if their euclidean distance is lower than $d$. Graph evolution between two consecutive time steps $t$ and $t+1$, is driven by two rules applied to each vertex $v \in V_t$ simultaneously. The first rule determines for a vertex $v \in V_t$ whether it is kept at step $t+1$ while the second rule concerns the possibility for a vertex $v \in V_t$ to create a new vertex in $V_{t+1}$ according to its degree.

▶ Remark. For generating new vertices by the second rule we had two possibilities. Either we choose, for each new vertex $w$ stemmed from a vertex $u$, a random position in a finite space, or, we choose a random position close to the position of $u$ with no constraint on space limits. We opted for the first option (the space is a unit 2D-torus), and we plan to analyze the differences with the second option.

▶ **Definition 4** (Conserved/Create/Removed/Duplicated nodes). *Let* $t \geqslant 0$ *and* $G = (G_t)$ *a D3G. Let* $u \in V_t$ *and* $v \in V_{t+1}$*, then*

- $u$ *is said to be a conserved node iff* $u \in V_t \cap V_{t+1}$*.*
- $u$ *is said to be removed iff* $u \in V_t - V_{t+1}$*.*
- $u$ *is said to be a creator/creating node iff* $deg(u) \in S_C$*.*
- $v$ *is said to be a created node iff* $v \in V_{t+1} - V_t$*.*
- $u$ *is said to duplicate iff it is both a conserved and a creator node.*

Once created, a node never change its position. Positions of created nodes do not depend on the creating nodes positions. The position of a created node is chosen randomly and uniformly over the unit-torus.

## 3 Theoretical Analysis

While the model is very simple, it presents a wide variety of dynamics and long-term evolution. According to $S_S$ and $S_C$ composition, several classes of dynamic behaviors have been identified. These classes have been defined by computing two measures: the evolution of the order of the graph, and the evolution of the Graph Nervousness $\mathcal{N}^G$. Results are reported in Table 1. A detailed analysis of each limit case is available in the report [5].

### 3.1 General Cases

General cases correspond to all cases for which both $S_C$ and $S_S$ are non empty sets and none of both sets are equal to $\mathbb{N}$. We classify all possible cases according to the tree represented on Figure 1.

The case $S_C = S_S$ composed of consecutive integers will be considered in Section 4. In the present section we consider the cases for which $S_C \neq S_S$.

**Table 1** Order, Nervousness evolution and sustainability property for the different cases. $n_t$ denotes the order of graph $G_t$, $\mathcal{N}^V(t)$ its vertices nervousness and $\mathcal{N}^G(t)$ the graph nervousness.

| $S_S$ \ $S_C$ | $\mathbb{N}$ | Finite set | $\emptyset$ |
|---|---|---|---|
| $\mathbb{N}$ | $\forall t, n_t = 2^t n_0$<br>$\forall t, \mathcal{N}^V(t) = 0.5$ | $\forall t, n_{t+1} \geqslant n_t$<br>$\forall t, 0 \leq \mathcal{N}^V(t) \leq 0.5$<br>$\lim_{n_t \to \infty} P(n_{t+1} > n_t) = 0$ | $\forall t, G_t = G_0$<br>$\forall t, \mathcal{N}^G(t) = (0,0)$ |
| | Sustainable | Asymptotically non sustainable | Non sustainable |
| Finite set | $\forall t, n_{t+1} \geqslant n_t$<br>$\forall t, 0.5 \leq \mathcal{N}^V(t) \leq 1$<br>$\lim_{n_t \to \infty} P(n_{t+1} > n_t) = 0$ | General cases<br>(see Section 3.1) | $\forall t, n_{t+1} \leqslant n_t$<br>$\lim_{t \to \infty} n = \text{constant}$<br>$\lim_{t \to \infty} \mathcal{N}^G(t) = (0,0)$ |
| | Sustainable | | Sustainable |
| $\emptyset$ | $\forall t, n_{t+1} = n_t$<br>$\forall t, \mathcal{N}^G(t) = (1,1)$ | $\forall t, n_{t+1} \leqslant n_t$<br>$\forall t, V_t \neq \emptyset \implies \mathcal{N}^G(t) = (1,1)$ | $\forall G_0, G_1 = (\emptyset, \emptyset)$ |
| | Sustainable | Depends on parameters | Non sustainable |

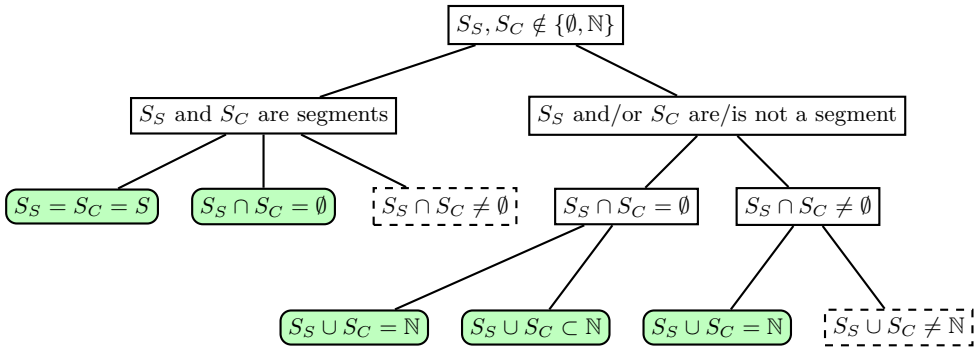



**Figure 1** Leaves of the tree represent the general cases. Rounded corners green boxes corresponds to cases for which results are presented in this section and in Section 4. Dashed boxes are cases not covered within this report.

- if $S_C \cap S_S = \emptyset$ and $S_C \cup S_S \subset \mathbb{N}$ then the order of the graph is non-increasing.
- if $S_C \cap S_S \neq \emptyset$ and $S_C \cup S_S = \mathbb{N}$ then the order of the graph is non-decreasing.
- If $S_C \cap S_S = \emptyset$ and $S_S \cup S_C = \mathbb{N}$, then $|V_t| = |V_0|$, the order of the graph is constant.

On the second time they are assumed to cover the whole set of natural integer numbers.

▶ **Theorem 5** (Disjoint sets). *Let $t \geqslant 0$ and $G_t = (V_t, E_t)$ a graph and $S_S$ and $S_C$ two sets of positive integers. If $S_S \cap S_C = \emptyset$, then the series $(|V_t|)_{t \geqslant 0}$ is decreasing.*

**Proof.** Let consider $(G_t)_{t \geqslant 0}$ a generated graph. Let $t \geqslant 0$ and $u$ be a vertex in $V_t$. Then, as $S_S \cap S_C = \emptyset$, the degree of node $u$ can't belong to both sets. It follows that vertex $u$ can't be both conserved and a creator. As this holds for every vertex in $V_t$, the order of generated snapshot graph is not increasing between two consecutive steps. ◀

▶ **Theorem 6** (Union set). *Let $S_S$ and $S_C$ subsets of $\mathbb{N}$. If $S_S \cup S_C = \mathbb{N}$, then the series $(|V_t|)_{t \geqslant 0}$ is increasing.*

**Proof.** The main argument here is the same used in the proof of theorem 5, except that the degree of every node in $V_t$ belongs to at least one of the two sets $S_S$ and $S_C$. Therefore, the order of generated snapshot graphs is not decreasing between two consecutive steps. ◀

### 3.1.1 Partition sets

In this section, $S_S$ and $S_C$ are considered to be a partition of $\mathbb{N}$. This means $S_S \cap S_C = \emptyset$ and $S_S \cup S_C = \mathbb{N}$. From theorems 5 and 6, one can say that for every graph $G_t = (V_t, E_t)$, the series $(|V_t|)_{t \geqslant 0}$ remains steady. Two cases rises from that situations:

- $S_S = \mathbb{N}$ **and** $S_C = \emptyset$: in that case the graph is constant ($\forall t, G_t = G_0$).
- $S_S = \emptyset$ **and** $S_C = \mathbb{N}$: the series of static graphs $(G_t)_{t \geqslant 0}$ is a series of independent random geometric graph with a constant number of nodes ($\forall t, n_t = n_0$).

## 4 Segments

This section focuses on the case $S_C = S_S = S$ where $S$ is a segment (i.e., an interval of consecutive integers).

### 4.1 Model and conjecture

In this section parameters $S_S$ and $S_C$ are limited to equal sets of consecutive integers. Both sets are such that $S_S = S_C = [m, M]$ (called segments), where $m, M \in \mathbb{N}^2$ and referred to as $S$ in the following. The evolution of graph order for different values of parameters $m$ and $M$ is investigated. Some statements and properties are theoretically and experimentally proved for the special case $S = \{0\}$. A relationship between graph order at a step $t+1$ and at step $t$ and an upper bound for $n_t$ ($t > 0$) are given. Then, a theoretical analysis of the general case is provided, and a new concept named sustainable interval is introduced. In the last part of this section, vertices nervousness of graphs is studied through experimentation. It is shown to be equal in average to $\frac{2}{3}$. The reason behind this particular value will be explained in this last part.

### 4.2 $S = \{0\}$

The case $S_S = S_C = S = \{0\}$ is considered in this section. The seed graph, $G_0$, is supposed to be a random geometric graph whose order is arbitrarily chosen. The main result about this case is an estimation of the mean value of graph order. An approximation for small values of the distance threshold $d$ is provided.

▶ **Theorem 7** (Expected value of graph order). *Let $S = \{0\}$, $d > 0$ and $G_0 = (V_0, E_0)$ such that there exists at least one node $u \in V_0$ being isolated (i.e., $deg(u) = 0$), then either the graph becomes empty, or the average number of conserved nodes is $l(d) = 1 - \frac{\log\left(\frac{\sqrt{1+4\alpha}-1}{2}\right)}{\log \alpha}$ with $\alpha = \frac{1}{1-p}$ and $p = p(d)$.*

**Proof.** Let $t \geqslant 1$. Two cases are to be discussed: the case of conserved vertices from step $t-1$ to step $t$ ($V_t \cap V_{t-1}$) and the case of created nodes at step $t$ ($V_t - V_{t-1}$). As the number of created nodes is the same as the number of conserved nodes from $t-1$ to $t$, we set $c_t = |V_t \cap V_{t-1}| = |V_t - V_{t-1}|$.

First let's study the number of conserved vertices from step $t$ to step $t+1$ among those conserved from step $t-1$ to step $t$. $c_{t+1}^{\text{conserved}}$ denotes this number. Let $u \in V_t \cap V_{t-1}$. The probability for $u$ to be conserved is the probability that its degree to created nodes remains equal to 0.

$$\deg(u) = \sum_{v \in V_t - V_{t-1}} X(u, v)$$

Let $v \in V_t - V_{t-1}$. As in the previous section, $X(u,v) \sim B(p)$ and $\deg(u) \sim B(c_t, p)$ as a sum of independent Bernoulli variables of same parameter $p$. $Y_t(u)$ denotes the event "$u$ is conserved at step $t+1$". The probability that $u$ survives is $P(Y_t(u)=1) = P(\deg(u)=0) = (1-p)^{c_t}$, thus: $Y_t(u) \sim B((1-p)^{c_t})$. Therefore, the number of conserved vertices at step $t+1$ among those conserved at step $t$ is:

$$c_{t+1}^{\text{conserved}} = \sum_{u \in V_t \cap V_{t-1}} Y_t(u)$$

As the position of created nodes are independent from themselves and from conserved vertices, $Y_t(u)$ are independent for all $u \in V_t \cap V_{t-1}$, $c_{t+1}^{\text{conserved}} \sim B(c_t, (1-p)^{c_t})$.

Let's study the number of conserved vertices among created nodes. $c_{t+1}^{\text{created}}$ denotes this number. Let $u \in V_t - V_{t-1}$. To study the degree of $u$, two cases must be studied. The first one is the number of connections between $u$ and all other created nodes (denoted as $\deg^C(u)$). The second one is the number of connections to already present nodes (denoted as $\deg^S(u)$). $\deg^C(u)$ and $\deg^S(u)$ can be obtained using the following formulas:

$$\deg^C(u) = \sum_{v \in V_t - V_{t-1}, u \neq v} X(u,v)$$
$$\deg^S(u) = \sum_{v \in V_t \cap V_{t-1}} X(u,v)$$

As the position of created points on the torus are independent one from the others, $\deg^C(u)$ is a sum of independent Bernoulli variables and therefore, $\deg^C(u) \sim B(c_t - 1, p)$. For $\deg^S(u)$, connections between a created node and an already present node are not independent from each other: knowing $u$ is connected to an already present node means it is close to it and as other conserved nodes are farther than $d$, it implies that $\deg^S(u)$ is not a sum of independent Bernoulli variables. However, as a first approximation, this quantity will be considered as a sum of independent Bernoulli variables.

Thus, the computation of the expectation of $c_{t+1} = c_{t+1}^{\text{conserved}} + c_{t+1}^{\text{created}}$ gives:

$$c_{t+1} = c_t(1-p)^{c_t} + c_t(1-p)^{2c_t - 1}$$

By looking for a limit to this series gives $l \geqslant 0$ satisfying:

$$l = l(1-p)^l + l(1-p)^{2l-1}$$

Solving this equation gives $l = 0$ or :

$$l = 1 - \frac{\log\left(\frac{\sqrt{1+4\alpha}-1}{2}\right)}{\log \alpha} \qquad \text{with } \alpha = \frac{1}{1-p}$$ ◀

Experiments have been run to see if this relationship holds.

▶ **Corollary 8.** *Let $d > 0$ and $l(d)$ as defined in the Theorem 7. Then for small values of $d$:*

$$l(d) \sim -\frac{\log\left(\frac{\sqrt{5}-1}{2}\right)}{\pi d^2} = \frac{\log \phi}{\pi d^2}$$

*where $\phi$ is the golden ratio $\left(\frac{1+\sqrt{5}}{2}\right)$.*

**Proof.** Let $d > 0$ be small. Thus, applying Taylor expansion gives $\frac{1}{1-\pi d^2} \sim 1 + \pi d^2$ and $\log\left(\frac{1}{1-\pi d^2}\right) \sim \pi d^2$. The numerator comes from $4 \cdot \frac{1}{1-\pi d^2} \simeq 4$. The golden ratio is obtained using operations on log and by noticing that $\frac{2}{\sqrt{5}-1} = \frac{2(\sqrt{5}+1)}{4} = \phi$, the golden ratio. Combining these results leads to the statement of the corollary. ◀

It is therefore possible to state that, in the case where $S = \{0\}$, it is possible to theoretically get an expectation of graph order as well as to get an upper bound for graph order depending on parameter $d$.

Experiments have been performed to see whether the expected value graph order holds. These experiments has been performed for different values of threshold $d$. A linear regression shows that the relationship holds with a $R^2$ of more than 0.99.

## 4.3 The general case

Now the focus is on $S = [m, M]$ for every $m$ and $M$ integers. The goal is to provide a tool aiming at stating, for given parameters $m, M$ and $d$, whether the graph is likely to be sustainable or not. This part mainly focuses on a simpler model. This model is studied as it helps understanding the evolution of graph order.

### 4.3.1 Study of graph evolution

In this Section we aim at estimating the evolution of the graph order during graph dynamics. However, in the D3G3 model, between two time steps, non-conserved nodes are removed from the graph and conserved nodes are located at the same position, which entails a remanent graph. This remanent graph induces a structure influencing the computation of graph order. More precisely, nodes that are about to be removed connected to conserved ones interfere in the probability that conserved nodes at time $t$ are still conserved at time $t+1$. This is linked to computing the degree of the neighbors of a node $u$ knowing the degree of node $u$. To our knowledge, this is a difficult question. For that purpose, a relaxed version of the D3G3 model is considered enabling analytical study of this evolution. In this model, conserved nodes are moved (i.e., their position are changed) such that obtained graph is a new random geometric graph at each step. We call this model "the redistributed model". This will help us proving the following theorem:

▶ **Theorem 9.** *Let $G = (G_t)$ be a dynamic graph obtained with the redistributed model, then at every step $t$, $\frac{n_{t+1}}{2} \sim B(n_t, p(S, d, n_t))$, where $p(S, d, n_t)$ is the probability that a node is conserved between step $t$ and $t+1$:*

$$p(S, d, n_t) = \sum_{k=m}^{M} \binom{n_t - 1}{k} p^k (1-p)^{n_t - 1 - k}$$

*Here, $p(d)$ refers to the probability for two different nodes to be connected (i.e., the probability that the distance between them is lower than or equal to $d$), which is, for $d \leqslant \frac{1}{2}$, $\pi d^2$.*

**Proof.** In the redistributed model, at time step $t$ a RGG $(G_t)$ is built. If the graph order at time $t$ is equal to $n_t$, the graph order at $t+1$ is equal to twice the number of surviving nodes at time $t$. As every node has an independent position in the torus, this probability is the same for all nodes. Let's denote it $p(S, d, n_t)$. Let $u \in V_t$. Then:

$$p(S, d, n_t) = P(\deg(u) \in S) = \sum_{k=m}^{M} P(\deg(u) = k) = \sum_{k=m}^{M} \binom{n_t - 1}{k} p^k (1-p)^{n_t - 1 - k} \quad (1)$$

Assuming one node is a conserved node, it does not affect the probability of conservation for other nodes. The number of conserved nodes can be computed summing independent Bernoulli's events of parameter $p(S, d, n_t)$. This gives $\frac{n_{t+1}}{2}$ follows a binomial distribution of parameter $n_t$ and $p(S, d, n_t)$. ◀

Computing expectation for a binomial distribution leads to an expectation for $n_{t+1}$ knowing $n_t$. Indeed, this expectation is $2n_t p(S, d, n_t)$. For a fixed set $S$, this provides a relationship between $n_t$ and $n_{t+1}$:

▶ **Definition 10** (Expectation of graph order). *Let $m$, $M$ and $d$ be parameters for the redistributed model. Let $G = (G_t)$ be an obtained graph with such parameters. Then, the expectation of graph order at step $t+1$ ($n_{t+1}$) knowing graph order at step $t$ ($n_t$) is $f_{S,d}(n_t)$ and satisfies $n_{t+1} = f_{S,d}(n_t) = 2n_t p(S, d, n_t)$, and then:*

$$\forall n \in \mathbb{N}, f_{S,d}(n) = 2np(S, d, n) \tag{2}$$

This quantity is referred to as the relationship in the sequel. Studying the relation for every value of $m$, $M$ and $d$ turns out to be a difficult problem. However some results may be conjectured. A first conjecture concerns the variations of the relationship:

▶ **Conjecture 11.** *Let $m, M$ and $d$ be parameters of the model. Let $S = [m, M]$ and $f_{S,d}$ the relationship as defined above. Then there exists $n_* \in \mathbb{N}$ such that $f_{S,d}$ is increasing on $[0, n_*]$ and decreasing on $[n_* + 1, +\infty[$.*

This conjecture is difficult to prove due to the sum involved in the computation of $f_{S,d}$. However, it is not necessary to study the relationship for all integers. It is possible to perform the study on a limited interval. This is the purpose of theorem 13 (below). But before proving this theorem, it is necessary to provide another formulae computing variations of $f_{S,d}$:

▶ **Lemma 12.** *Let $m$, $M$ and $d$ be parameters of the model. Let $\Delta f_{S,d}$ defined as the variation of $f_{S,d}$: for $n \in \mathbb{N}, \Delta f_{S,d}(n) = f_{S,d}(n+1) - f_{S,d}(n)$. Then:*

$$\forall n \in \mathbb{N}, \Delta f_{S,d}(n) = 2\sum_{k=m}^{M} (k+1)\binom{n}{k} p^k (1-p)^{n-1-k} \left(1 - \frac{n+1}{k+1}p\right)$$

**Proof.** Let $m$, $M$ and $d$ be parameters of the model. Let $n$ a be non-negative integer. This proof only focuses on the terms of the sum of $\Delta f_{S,d}$:

$$\begin{aligned}\Delta f_{S,d}(n) &= 2\left(\sum_{k=m}^{M} (n+1)\binom{n}{k} p^k (1-p)^{n-k} - n\binom{n-1}{k} p^k (1-p)^{n-1-k}\right) \\ &= 2\sum_{k=m}^{M} p^k (1-p)^{n-1-k} \left((n+1)\binom{n}{k}(1-p) - \binom{n-1}{k}\right)\end{aligned}$$

Let $k \in \mathbb{N}$ such that $m \leqslant k \leqslant M$. Every term of the sum of $\Delta f_{S,d}$ can be expressed as follow only using results on binomial coefficients:

$$\Delta f_{S,d}(n) = 2\sum_{k=m}^{M} (k+1)\binom{n}{k} p^k (1-p)^{n-1-k} \left(1 - \frac{n+1}{k+1}p\right)$$ ◀

It is now possible to state the following theorem about variations of $f_{S,d}$:

▶ **Theorem 13.** *Let $m, M$ and $d$ be the parameters of the model. Let $S = [m, M]$ and $f_{S,d}$ the relationship as defined above. Let $p = p(d)$ be the probability for two different nodes to be connected. Then, $f_{S,d}$ is increasing between 0 and $\frac{m+1}{p} - 1$ and decreasing from $\frac{M+1}{p} - 1$ to infinity.*

**Proof.** The goal is to prove that $\Delta f_{S,d}(n)$ is positive for $n < \frac{m+1}{p} - 1$ and negative for $n > \frac{M+1}{p} - 1$. To understand this, $\Delta f_{S,d}(n)$ can be rewritten as shown in lemma 12. It is sufficient to notice that, for all $k \in S$, the sign of every single term of the sum is the sign of $\left(1 - \frac{n+1}{k+1}p\right)$. For fixed $k$, the term is positive if and only if $n$ is lower than $\frac{k+1}{p} - 1$. As this last term is an increasing function of $k$, all terms of the sum are therefore positive if $n$ is lower than $\frac{m+1}{p} - 1$ and negative if $n$ is greater than $\frac{M+1}{p} - 1$. Hence, the relationship is increasing from 0 to $\frac{m+1}{p} - 1$ and decreasing from $\frac{M+1}{p} - 1$ to infinity. ◀

Thanks to theorem 13, conjecture 11 is proved for intervals $[0, x_m]$ and $[x_M, \infty[$ for $x_m = \frac{m+1}{p} - 1$ and $x_M = \frac{M+1}{p} - 1$. At this stage, quantifying more precisely the evolution of the graph order is not achievable. However, a study of the fixed points of $f_{S,d}$ enables to draw some conclusion about generated graphs sustainability.

### 4.3.2 Graph evolution and sustainability

First note that knowing the variations of $f_{S,d}$ is not enough to deal with graphs sustainability. Indeed, as claimed by the following theorem, big graphs are not sustainable.

▶ **Theorem 14** (Non-sustainability of big graphs)**.** *Let $m$, $M$ and $d$ be parameters of the model. Let $f_{S,d}$ be the relationship. Then, there exists $N > 0$ such that for all $n > N, f_{S,d}(n) < 1$.*

**Proof.** For this proof, it is sufficient to prove that $f_{S,d}(n) \to 0$ when $n \to +\infty$. To do so, $f_{S,d}(n)$ can be rewritten as follow:

$$f_{S,d}(n) = 2n \left( \sum_{k=m}^{M} \binom{n-1}{k} p^k (1-p)^{n-k-1} \right)$$

From theorem 24, the sum tends toward 0 as the product of a polynomial and an exponential, therefore, $f_{S,d}(n)$ is also tending toward 0. ◀

This theorem says that there always exists a graph order limit such that graphs whose order are greater than this limit are likely to become empty. Therefore, it is not possible to obtain sustainable graphs with a large amount of nodes.

A new mathematical concept is now introduced aiming at classifying parameters into three classes. This concept is referred to fixed point and is defined as follow:

▶ **Definition 15** (Fixed Point)**.** *Let $m$, $M$ and $d$ be parameters of the model. A fixed point for the relationship $f_{S,d}$ is an non-negative integer $n$ such that:*

$$\begin{cases} & f_{S,d}(n) \leqslant n \text{ and } f_{S,d}(n+1) > n+1 \\ \text{or} & f_{S,d}(n) \geqslant n \text{ and } f_{S,d}(n+1) < n+1 \end{cases}$$

Such fixed points characterize variation of graph order. Indeed, graph of order $n$ for $n$ taken between two consecutive fixed points is either always decreasing or increasing. From experiment performed on the redistributed model as well as on D3G3, three different cases appear and are conjectured as follow:

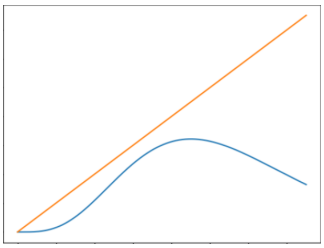
▶ **Conjecture 16.** *For all $m$, $M$ and $d$ being parameters of the model, the relationship $f_{S,d}$ has either one, two or three fixed points.*

This conjecture is the main tool aiming at studying sustainability in the segment case. Indeed, in the three different cases, it is possible to answer whether a given set of parameters is sustainable or not. However, their is no characterisation about parameters value that may help founding which case parameters lead to. The only one claim that can be made is that $d$ does have an influence on this case.

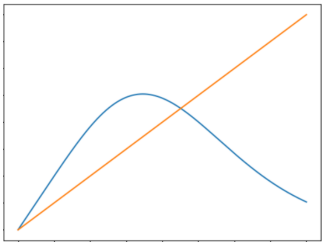
The conjecture 16 is assumed in this subsection. This section aims at stating about sustainability in the three different cases. This is illustrated by a description of the behavior of the relationship $f_{S,d}$ in every case.

#### 4.3.2.1 One fixed point

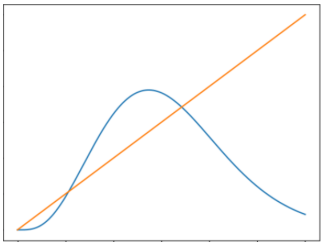
First let's consider the case where the relationship has only one fixed point. When it has only one fixed point, this point is 0. This comes from $f_{S,d}(0) = 0$. Moreover, for all $n$, $f_{S,d}(n) < n$. As for a snapshot graph of order $n_t$ at step $t$, $f_{S,d}(n_t)$ gives the expectation value of $n_{t+1}$ at step $t + 1$. Graph orders of generated graphs are decreasing in average. Graphs obtained in this case are therefore not sustainable.



One fixed point.



Two fixed points.



Three fixed points.

#### 4.3.2.2 Two fixed points

For the two fixed points case, 0 is also a fixed point. The argument is also because $f_{S,d}(0) = 0$. The other one is greater than zero. The case where $f_{S,d}$ has two fixed points is assumed to happen if and only if $0 \in S$ and is stated in the following conjecture.

▶ **Conjecture 17** (Characterisation of the two fixed points)**.** *The relationship $f_{S,d}$ has two fixed points if and only if $m = 0$.*

An argument is that a snapshot graph with one vertex becomes empty if and only if $0 \notin S$, that is $m = 0$. Moreover, for $n = 0$, the first term of the sum defining $f_{S,d}$ is equal to 1 so $f_{S,d}(0) = 2$. Graphs generated in such configurations are therefore sustainable as long as their graph order does not exceed a limit. This limit is a consequence of theorem 14. In this case, graphs whose order exceeds the limit are likely to become empty.

#### 4.3.2.3 Three fixed points

For the last case, the goal is to show that graph order is likely to remain bounded. Deeply looking at this case raises the question of values of graph order for which the size is not too large and not too small so that it does not collapse. For that purpose we define an interval, called *sustainable interval*, such that, if the graph order remains within that interval, this ensures the persistence of the graph. This sustainable interval is considered as a tool to study graph sustainability. It concerns expectation of graph order evolution through time. It

says that if the image of the function $f_{S,d}$ for all integers within the interval does not exceed the upper bound, then the graph is likely not to collapse. Let's define more precisely this concept:

▶ **Definition 18** (sustainable interval). *Let $m$, $M$ and $d$ be parameters of the model. Let consider $f_{S,d}$ set such that it has three fixed points. Let $N_m$ be the first positive fixed point and $N'_m$ the smallest integer greater than $N_m$ such that $f_{S,d}(N'_m) \geqslant N_m$ and $f_{S,d}(N'_m + 1) < N_m$. The sustainable interval associated to $m$, $M$ and $d$ is defined as the interval $[N_m, N'_m]$.*

Such an interval satisfies a property about the values $f_{S,d}$ takes when it is restricted to it:

▶ **Theorem 19** (Sustainability in the sustainable interval). *Let $m$, $M$ and $d$ be parameters of the model. Let assume the relationship $f_{S,d}$ has three fixed points and that $[N_m, N'_m]$ is its associated sustainable interval. If the relationship does not exceed $N'_m$, then the relationship satisfies:*

$$\forall n \in [N_m, N'_m], f_{S,d}(n) \in [N_m, N'_m]$$

**Proof.** The lower bound of the interval comes comes from definition of the sustainable interval. The upper bound is a hypothesis of the theorem. ◀

Main interpretation of that theorem is graphs are sustainable in probability in the sustainable interval if and only if there are no values of $f_{S,d}$ that exceed the upper bound of the sustainable interval.

The following paragraphs provide arguments aiming at obtaining the sustainable interval. They also provide arguments to check whether the relationship exceeds the upper bound of the interval. The theorem 13 clearly gives bounds to find out the maximum of the relationship $f_{S,d}$. Three algorithms are sufficient to answer both questions: an algorithm to compute the argument of the maximum of the relationship $f_{S,d}$, an algorithm to find its fixed point between 0 and the argument of the maximum and an algorithm to solve $f_{S,d}(n) = y$ for $n$ greater than the argument of the maximum and $y > 0$ lower than or equal to the maximum. In the following, these algorithms are first implemented. It is then explained how to use them to answer questions about the sustainable interval.

**The argument maximum:** To compute the argument maximum of the relationship, it is sufficient to study $f_{S,d}$ on the interval $[x_m, x_M]$ for $x_m$ and $x_M$ as defined above. This is a consequence of theorem 13. Let's denote it $N_*$.

**The first positive fixed point:** To find the fixed point of $f_{S,d}$ mentioned in the definition of the sustainable interval, it is sufficient to compute the argument maximum of it. The previous algorithm answers this question. Then, as the relationship is increasing from 0 to $N_*$, it is sufficient to iterate and find an integer $n$ such that $f_{S,d}(n) \leqslant n$ and $f_{S,d}(n+1) > n+1$.

**The solution of the equation:** For the last algorithm, the goal is to find an integer $n$ such that $n$ is greater than $N_*$ of $f_{S,d}$, $f_{S,d}(n) \geqslant y$ and $f_{S,d}(n) < y$, for a fixed $y$ which is assumed to be positive and lower than the maximum of $f_{S,d}$.

From these algorithms it is possible to implement algorithms stating the existence of the sustainable interval and its bounds. For the existence or not of the sustainable interval, it is sufficient to check whether the maximum of the relationship is greater than its argument. This comes from that sustainable interval exists if and only if there are values of the relationship

that exceed their argument. As the relationship is increasing from 0 to $f_{S,d}(N_*)$, then sustainable interval exists if and only if $f_{S,d}(N_*) > N_*$. For computing the sustainable interval boundaries, it is sufficient to know the value of the first fixed point $N_m$ (as it provides the lower bound) and to solve the equation $f_{S,d}(x) = N_m$ as finding the corresponding $x$ to this equation provides the upper bound $(N'_m)$. The existence of $N'_m$ is ensured by theorem 14.

## 4.4 Vertex nervousness

The goal is to highlight a characterization aspect of the segment family using the vertex nervousness metric. As edge nervousness will not be studied for that case, vertex nervousness will be referred to as nervousness in this section. As in this particular configuration, survivors are the same as created nodes, it is possible to state particular results about the value of nervousness:

▶ **Theorem 20.** *Let $S$ be a segment set of non-negative integer and $d \in (0, \frac{\sqrt{2}}{2})$. Let $G$ be a generated graph of order $n_t$ at step $t$ and number of survivor from step $t$ to step $t+1$ referred to as $s_t$. Then:*

$$\mathcal{N}_t^v = \frac{n_t}{n_t + s_t}$$

**Proof.** To prove this result, it is sufficient to notice that $n_{t+1} = 2s_t$, as $S_S = S_C$, which means the number of survivors is the same as the number of created nodes. Thus, applying some basic result about set sizes and noticing that $s_t = |V_t \cap V_{t+1}|$, leads to:

$$|V_t \cup V_{t+1}| = n_t + n_{t+1} - s_t = n_t + s_t$$
$$|V_t \triangle V_{t+1}| = n_t + n_{t+1} - 2|V_t \cap V_{t+1}| = n_t$$

It follows that vertex nervousness is well equal to $\frac{n_t}{n_t+s_t}$. ◀

Result about the nervousness observed in generated graphs parameterized with a segment set $S$ is stated in the following conjecture:

▶ **Conjecture 21.** *Let $m, M \in \mathbb{N}$. Let $S = [m, M]$ and $d > 0$ be parameters of the generator and let $G = (G_t)_{t \leqslant 0}$ be a generated graph. Then the vertex nervousness is in average equal to $\frac{2}{3}$.*

Although this conjecture has not been proved theoretically, experimentation have been performed. They all highlight this conjecture telling that the average nervousness of generated graphs is roughly equal to $\frac{2}{3}$. Results of this experimentation are gathered on picture 2. A possible interpretation of this conjecture and performed experimentation relies on the result stated in theorem 20 and on results from last part. Indeed, if vertex nervousness is close to $\frac{2}{3}$, it means $s_t \simeq \frac{n_t}{2}$. Then, as $n_{t+1} = 2s_t$, it comes $n_{t+1} \simeq n_t$, which means that graph order is close to a fixed point of the relationship $f_{S,d}$ mentioned in the previous section.

# 5 Conclusion

This paper shows our first investigations in the study of dynamic graph generators. This work concerns a simple generator. As a reminder, the model is parameterized through three variables: a connection threshold $d$ aiming at connecting all points closer than a distance $d$ and two sets $S_S$ and $S_C$ containing non-negative integers. The first one aims at deciding whether a node is kept between two consecutive steps and the second one whether a node is at the origin of a new node at the very next step. Several non-trivial properties are shown

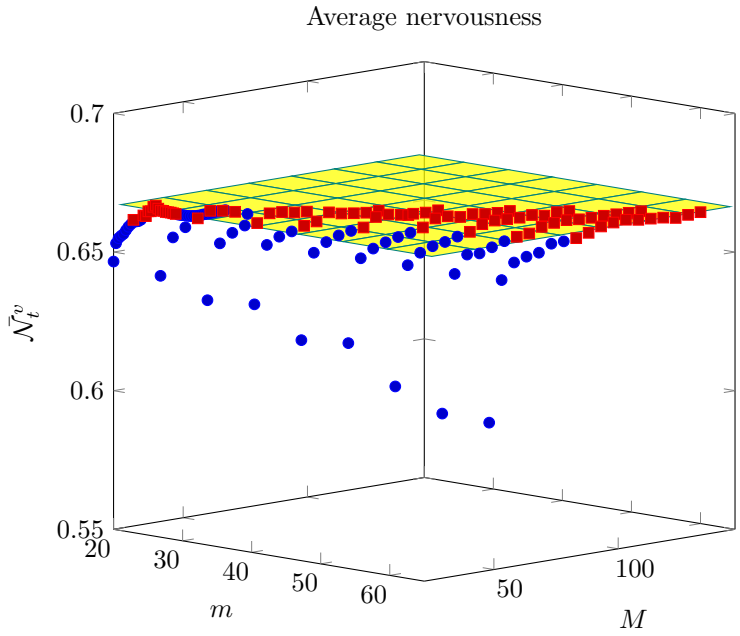



**Figure 2** Mean value of nervousness got from experimentation. Points represent the average over 20 runs and 30000 time steps for a single $m$ and $M$. The yellow surface is the plan of equation $z = \frac{2}{3}$. For all these parameters, $d$ is set to 0.05. Red points represent nervousness of value greater than $\frac{2}{3}$. Blue points represent nervousness of value lower than $\frac{2}{3}$.

about the model. All these properties concern products of the generator. The generator, for a single configuration, produces a family of graphs and not a single graph. Properties are therefore about the whole family of graphs the generator provides for a single configuration. All these properties shown try to answer a single question. This question concerns graph *sustainability*. It is defined as the property, for a given graph obtained with a given seed graph and evolving rules, that the graph becomes neither empty after a finite number of steps nor periodic. Defining this concept for this model is not simple since the evolving rules are not deterministic. It involves probabilistic computations and therefore questions about a possible threshold for which the graph is said to be sustained if the probability of the emptiness of the graph is greater than this threshold. Here the focus has been made on two different metrics, graph order evolution and vertex nervousness, the second one being a renaming of the Jaccard distance metric. Different values of the parameters have been studied, but it has not been possible to try them all as the amount of possible cases is far too big. Cases for which properties have been shown are limit cases, the general case and a very specific case referred to as "segments". Limit cases have led to a first classification when at least one of the two parameter sets is either empty or contains all non-negative integers. General cases highlights some properties for specific values of the two sets. Finally, the case where both sets are equal and contains consecutive non-negative integers has been studied. These sets are called segments. It has revealed theoretical difficulties, especially when computing graph order between consecutive steps. This has led to the creation of a new tool named the "sustainable interval". This tool aims at estimating bounds that frames graph order even though it is not always reliable as probabilities are involved. This last study is only about equal sets. For further studies, the case where both sets are segments but not equal seems relevant as it does not change too much from the segment case.

## References


1 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999. doi:10.1126/science.286.5439.509.

2 S Boccaletti, V Latora, Y Moreno, M Chavez, and D Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(4-5):175–308, 2006.

3 Marián Boguñá, Dmitri Krioukov, and K. C. Claffy. Navigability of complex networks. *Nature Physics*, 5(1):74–80, January 2009. Number: 1 Publisher: Nature Publishing Group. doi:10.1038/nphys1130.

4 Béla Bollobás. A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs. *European Journal of Combinatorics*, 1(4):311–316, December 1980. doi:10.1016/S0195-6698(80)80030-8.

5 Vincent Bridonneau, Frédéric Guinand, and Yoann Pigné. Dynamic Graphs Generators Analysis : an Illustrative Case Study. Technical report, LITIS, Le Havre Normandie University, December 2022. URL: https://hal.science/hal-03910386.

6 Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding Time of Edge-Markovian Evolving Graphs. *SIAM Journal on Discrete Mathematics*, 24(4):1694–1712, January 2010. doi:10.1137/090756053.

7 Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

8 Afonso Ferreira and Laurent Viennot. *A Note on Models, Algorithms, and Data Structures for Dynamic Communication Networks*. report, INRIA, 2002. URL: https://hal.inria.fr/inria-00072185.

9 Guillermo García-Pérez, M. Ángeles Serrano, and Marián Boguñá. Soft communities in similarity space. *Journal of Statistical Physics*, 173(3):775–782, 2018. doi:10.1007/s10955-018-2084-z.

10 K.-I. Goh and A.-L. Barabási. Burstiness and memory in complex systems. *EPL (Europhysics Letters)*, 81(4):48002, January 2008. doi:10.1209/0295-5075/81/48002.

11 Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912. doi:10.1111/j.1469-8137.1912.tb05611.x.

12 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.

13 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, September 2010. Publisher: American Physical Society. doi:10.1103/PhysRevE.82.036106.

14 Per Lundberg, Esa Ranta, Jörgen Ripa, and Veijo Kaitala. Population variability in space and time. *Trends in Ecology & Evolution*, 15(11):460–464, November 2000. doi:10.1016/s0169-5347(00)01981-9.

15 Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

16 Alessandro Muscoloni and Carlo Vittorio Cannistraci. A nonuniform popularity-similarity optimization (nPSO) model to efficiently generate realistic complex networks with communities. *New Journal of Physics*, 20(5):052002, May 2018. doi:10.1088/1367-2630/aac06f.

17 Fragkiskos Papadopoulos, Maksim Kitsak, M. Ángeles Serrano, Marián Boguñá, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489(7417):537–540, September 2012. doi:10.1038/nature11459.

18 Nicola Santoro, Walter Quattrociocchi, Paola Flocchini, Arnaud Casteigts, and Frederic Amblard. Time-Varying Graphs and Social Network Analysis: Temporal Indicators and Metrics, February 2011. arXiv:1102.0629 [physics]. doi:10.48550/arXiv.1102.0629.

19 M. Ángeles Serrano, Dmitri Krioukov, and Marián Boguñá. Self-similarity of complex networks and hidden metric spaces. *Phys. Rev. Lett.*, 100:078701, February 2008. doi:10.1103/PhysRevLett.100.078701.

20 Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. In *Social networks*, pages 179–197. Elsevier, 1977.

21 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, June 1998. doi:10.1038/30918.

22 Konstantin Zuev, Marián Boguñá, Ginestra Bianconi, and Dmitri Krioukov. Emergence of Soft Communities from Geometric Preferential Attachment. *Scientific Reports*, 5(1):9421, August 2015. doi:10.1038/srep09421.


# A Appendices

## A.1 Binomial coefficient and Binomial distribution

This section aims at providing results about binomial coefficient and binomial distribution. From now the objective of the two following theorems is to provide asymptotic equivalent of expressions involving a binomial coefficient. The first theorem gives an asymptotic equivalent of $\binom{n}{k}$:

▶ **Theorem 22** (Asymptotic analysis of binomial coefficient)**.** *Let $k$ and $n$ be non-negative integers such that $k \leqslant n$ and $k$ does not depend on $n$. Then as $n$ tends to infinity, the following holds*

$$\binom{n}{k} \sim \frac{n^k}{k!}$$

**Proof.** Let $k$ and $n$ as in the above statement. Then, it is sufficient to rewrite the binomial coefficient as follow:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{1}{k!}\prod_{i=0}^{k-1}(n-i) \sim \frac{n^k}{k!} \quad \text{(Asymptotic equivalent of a polynomial)}$$ ◀

The following theorem provides an equivalent of the mass function of a binomial distribution:

▶ **Theorem 23** (Asymptotic analysis of binomial distribution)**.** *Let $k$ and $n$ be non-negative integers such that $k$ does not depend on $n$. Let $x \in (0,1)$. Then the following limit holds:*

$$\lim_{n\to+\infty} \binom{n}{k} x^k (1-x)^{n-k} = 0$$

*Moreover the following equivalent can be expressed:*

$$\binom{n}{k} x^k (1-x)^{n-k} \sim \frac{1}{k!}\left(\frac{x}{1-x}\right)^k n^k (1-p)^n$$

**Proof.** Let $k$, $n$ and $x$ as defined in the above statement. As stated in theorem 22, an asymptotic equivalent of $\binom{n}{k}$ is $\frac{1}{k!} \times n^k$. Therefore, the following holds:

$$\binom{n}{k} x^k (1-p)^{n-k} = \left(\frac{x}{1-x}\right)^k \binom{n}{k} x^n \sim \frac{1}{k!}\left(\frac{x}{1-x}\right)^k n^k x^n$$

To conclude it is sufficient to notice that $n^k x^n$ tends toward 0 as $n$ tends to infinity (due to $x \in (0,1)$). ◀

▶ **Theorem 24.** *Let $A \subset \mathbb{N}$ be a finite set of non-negative integers. Let $n$ be a non-negative integer and $x \in (0,1)$. Then the following holds*

$$\lim_{n\to+\infty} \left(\sum_{k\in A} \binom{n}{k} x^k (1-x)^{n-k}\right) = 0$$

**Proof.** As set $A$ is finite, the sum in the statement has a finite number of terms. Let denote $a = |A|$ and $M = \max A$. For values of $n$ such that $n \geqslant 2M$, the following inequality holds:

$$\forall k \leqslant M, \binom{n}{k} \leqslant \binom{n}{M}$$

Moreover, as $1-x < 1$, $y \longmapsto (1-x)^{n-y}$ is increasing. Therefore, for all $k \leqslant M$, $(1-x)^{n-k} \leqslant (1-x)^{n-M}$. It is thus possible to get the following inequality for all $k \leqslant M$:

$$0 \leqslant \binom{n}{k}(1-x)^{n-k}x^k \leqslant \binom{n}{M}(1-p)^{n-M}x^k \leqslant \binom{n}{M}(1-p)^{n-M}$$

Noticing the sum is composed of $a$ elements, it can be bounded as follow

$$0 \leqslant \sum_{k\in A}\binom{n}{k}x^k(1-x)^{n-k} \leqslant a\binom{n}{M}(1-x)^{n-M}$$

As $M$ is fixed an equivalent to right term of the previous inequality as $n$ grows to infinity is:

$$a\binom{n}{M}(1-x)^{n-M} \sim \frac{a}{(1-x)^M}\frac{n^M}{M!e^M}(1-x)^n$$

As $n^M x^n$ tends toward 0, the right term of the equivalent tends toward 0 too. Moreover, the sum is composed of positive elements. Applying the squeeze theorem leads to the wanted limit. ◀

▶ **Theorem 25.** *Let $k$ and $n$ be two non-negative integers. Let $x \in (0,1)$. Then, the following holds:*

$$(n+1)\binom{n}{k}(1-p) - n\binom{n-1}{k} = (k+1)\binom{n}{k}\left(1-p\frac{n+1}{k+1}\right)$$

**Proof.** The above statement can be proved with the following equations:

$$\begin{aligned}
(n+1)\binom{n}{k}(1-p) - n\binom{n-1}{k} &= (k+1)\binom{n+1}{k+1}(1-p) - (k+1)\binom{n}{k+1} \\
&= (k+1)\left(\binom{n+1}{k+1} - p\binom{n+1}{k+1} - \binom{n}{k+1}\right) \\
&= (k+1)\left(\binom{n}{k} - p\binom{n+1}{k+1}\right) \\
&= (k+1)\left(\binom{n}{k} - p\frac{n+1}{k+1}\binom{n}{k}\right) \\
&= (k+1)\binom{n}{k}\left(1-p\frac{n+1}{k+1}\right)
\end{aligned}$$

◀

# Complexity of the Temporal Shortest Path Interdiction Problem

**Jan Boeckmann**
TUM Campus Straubing for Biotechnology and Sustainability,
Hochschule Weihenstephan-Triesdorf, Germany

**Clemens Thielen**
TUM Campus Straubing for Biotechnology and Sustainability,
Hochschule Weihenstephan-Triesdorf, Germany
Department of Mathematics, School of Computation, Information and Technology,
Technische Universität München, Germany

**Alina Wittmann**
Department of Mathematics, School of Computation, Information and Technology,
Technische Universität München, Germany

## Abstract

In the shortest path interdiction problem, an interdictor aims to remove arcs of total cost at most a given budget from a directed graph with given arc costs and traversal times such that the length of a shortest $s$-$t$-path is maximized. For static graphs, this problem is known to be strongly $\mathcal{NP}$-hard, and it has received considerable attention in the literature.

While the shortest path problem is one of the most fundamental and well-studied problems also for temporal graphs, the shortest path interdiction problem has not yet been formally studied on temporal graphs, where common definitions of a "shortest path" include: *latest start path* (path with maximum start time), *earliest arrival path* (path with minimum arrival time), *shortest duration path* (path with minimum traveling time including waiting times at nodes), and *shortest traversal path* (path with minimum traveling time *not* including waiting times at nodes).

In this paper, we analyze the complexity of the shortest path interdiction problem on temporal graphs with respect to all four definitions of a shortest path mentioned above. Even though the shortest path interdiction problem on static graphs is known to be strongly $\mathcal{NP}$-hard, we show that the latest start and the earliest arrival path interdiction problems on temporal graphs are polynomial-time solvable. For the shortest duration and shortest traversal path interdiction problems, however, we show strong $\mathcal{NP}$-hardness, but we obtain polynomial-time algorithms for these problems on extension-parallel temporal graphs.

**2012 ACM Subject Classification** Theory of computation → Shortest paths; Theory of computation → Dynamic graph algorithms; Mathematics of computing → Paths and connectivity problems

**Keywords and phrases** Temporal Graphs, Interdiction Problems, Complexity, Shortest Paths, Most Vital Arcs

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.9

## 1 Introduction

Not least because of its great applicability to a wide range of real-world problems, the shortest $s$-$t$-path problem is undeniably one of the most central and well-studied problems in graph theory and network optimization. On static graphs, where the graph is not subject to change over time, efficient algorithms to solve the shortest path problem are known. The assumption of a graph not changing over time, however, is often too restrictive when modeling real-world problems such as the spread of the virus during the COVID-19 pandemic. In such settings, the concept of *temporal graphs*, where arcs are only available at certain times, allows for more realistic models (see, e.g., [4, 17]) and has recently attracted the interest of researchers in algorithmic network optimization (see, e.g., [1, 26, 27]).

© Jan Boeckmann, Clemens Thielen, and Alina Wittmann;
licensed under Creative Commons License CC-BY 4.0



Another topic that has become increasingly important in times of uncertainty are *interdiction problems*, where an interdictor aims to remove arcs of total cost at most a given budget from a (directed or undirected) graph or network such that the optimal objective value of an optimization problem on the resulting graph or network is maximized (in case of a minimization problem) or minimized (in case of a maximization problem). An important example of a highly relevant interdiction problem is the *shortest path interdiction problem*, where arcs are to be removed from a graph subject to a given budget such that the length of a shortest $s$-$t$-path for two given nodes $s$ and $t$ is maximized. Another example of a well-studied interdiction problem is the *network flow interdiction problem*, where arcs are to be removed from a network subject to a given budget such that the value of a maximum $s$-$t$-flow between two given nodes $s$ and $t$ is minimized [33, 37]. This problem is known to be strongly $\mathcal{NP}$-hard and several results about its approximability have been obtained [8, 11, 15].

In this paper, we investigate the *temporal shortest path interdiction problem*, where the aim is to remove arcs from a directed temporal graph such that the length of a shortest path from a node $s$ to another node $t$ is maximized. As the length of a path in a temporal graph can be interpreted in various different ways, we investigate four common versions of the temporal shortest path interdiction problem. We show that two of these versions are polynomial-time solvable, while the other two are strongly $\mathcal{NP}$-hard.

## 1.1 Previous Work

The following paragraphs summarize the state-of-the-art concerning shortest path problems on temporal graphs, the (static) shortest path interdiction problem, and related interdiction problems on temporal graphs.

We start with an overview of the literature about shortest path problems on temporal graphs. The model of a temporal graph used in this paper (and, e.g., in [10, 38]) is sometimes also referred to as a *scheduled network* [7] or a *point-availability time-dependent network* [9]. Here, each temporal arc $r$ can only be entered at a given start time $\tau(r)$ and it takes $\lambda(r)$ units of time to traverse the arc, which leads to an arrival time of $\tau(r) + \lambda(r)$ at the end node of the arc. In this model, four different definitions of a "shortest path" between two nodes $s$ and $t$ are considered (see [38]):

- *reverse-foremost* or *latest start* path, which is an $s$-$t$-path with maximum start time of the first arc in the path,
- *foremost* or *earliest arrival* path, which is an $s$-$t$-path with minimum arrival time of the last arc in the path,
- *shortest duration* path, which is an $s$-$t$-path with minimum total traveling time including waiting times at the nodes,
- *shortest traversal* path, which is an $s$-$t$-path with minimum total traveling time *not* including waiting times at the nodes.

For each of the four definitions, the corresponding temporal shortest path problem can be solved efficiently, i.e., a shortest path can be computed in polynomial time [5, 10, 38].

A different definition of temporal graphs is considered, e.g, in [27], where a wide range of well-studied graph problems is investigated on temporal graphs. This definition can be interpreted as the special case of the previous definition obtained when all traversal times are zero.[1] Biobjective versions of temporal shortest path problems are considered in [9, 30, 31].

[1] This implies that all polynomial-time solvability results presented here can immediately be transferred to the definition used in [27]. For our hardness results, we point out explicitly whether they can be transferred to this definition of temporal graphs.

A definition that allows for continuous availability of arcs in a temporal graph as well as a time dependency of an arc's traversal time is provided in [13]. This definition can be seen as a generalization of the definition from [10, 38] used here. However, due to the definition's large generality, it does not allow for a finite encoding of temporal graphs without imposing further assumptions, so classical techniques of complexity analysis cannot be applied for the most general form of this definition. A natural finite encoding is possible, e.g., if each arc is restricted to be present over a time interval, i.e., it can be entered at any time between two specified points in time. Even for this special case of the definition in [13], it is shown in Section 4.1 that deciding whether two nodes $s$ and $t$ can be separated by removing no more than $B$ arcs from the graph is already strongly $\mathcal{NP}$-hard.

Next, the literature about the shortest path interdiction problem on static graphs is summarized. To explicitly distinguish between the problem on *static* graphs and the problem on *temporal* graphs, we refer to the shortest path interdiction problem on static graphs as the *static* shortest path interdiction problem (S-SP-IP) in the following. This problem is also referred to as the *most vital arcs problem* in the literature [2]. S-SP-IP is one of the most-studied network interdiction problems and a vast amount of literature exists on the problem. A detailed overview is provided in [35]. Concerning the complexity of S-SP-IP, the first proof of weak $\mathcal{NP}$-hardness is provided in [2]. This result is extended in [3], where it is shown that S-SP-IP is strongly $\mathcal{NP}$-hard even on acyclic graphs and for the special case of unit arc lengths and removal costs. This result is further extended in [24], where it is shown that it is $\mathcal{NP}$-hard to approximate S-SP-IP within any factor $\alpha < 2$. Indeed, it is still an open question whether any non-trivial approximation algorithms exist for S-SP-IP. Variations of S-SP-IP considering online settings, randomized interdiction strategies, or multiple objectives have recently been studied, e.g., in [12, 21, 34].

While, to the best of our knowledge, the complexity of the shortest path interdiction problem has not been formally investigated on temporal graphs, a polynomial-time algorithm that decides whether there exist $k$ arc-disjoint temporal $s$-$t$-paths is presented in [7]. They further show that it can be decided in polynomial time whether there exists a temporal $s$-$t$-path arriving before a given arrival time even if up to $k$ arcs are removed, which implicitly solves the temporal earliest arrival path interdiction problem for unit removal costs. However, it is not clear whether the algorithm can be extended to the case in which arcs can have different removal costs.

Further, related reachability interdiction problems on temporal graphs are studied in [16, 17, 18, 29]. Here, the goal is to minimize (or maximize in some cases) the number of nodes reachable from a single node or a set of nodes in a temporal graph by either removing arcs, delaying start times, or changing the order of start times. While the vast majority of studied problems turn out to be $\mathcal{NP}$-hard even under severe restrictions, only a few special cases are shown to be polynomial-time solvable. Moreover, the problem of separating two given nodes by removing nodes from a temporal graph is considered for various settings in [20, 22, 23, 25, 28, 39]. Again, most of the problems are $\mathcal{NP}$-hard, while some polynomial-time solvability results – mostly for specific classes of graphs – are shown.

## 1.2 Our Contribution

We analyze the complexity of the shortest path interdiction problem on temporal graphs with respect to all four definitions of a shortest path considered in [38]. Even though S-SP-IP is known to be strongly $\mathcal{NP}$-hard, it is found that polynomial-time algorithms for the latest start and the earliest arrival path interdiction problem on temporal graphs exist. These algorithms exploit the fact that, for these versions of the problem, the objective value of a

path only depends on either the first or on the last arc in the path (but not on both). For the shortest duration and shortest traversal path interdiction problem, where both the first and the last arc in a path (and the amount of time spend waiting at nodes in the former case) are relevant for its length, however, we show strong $\mathcal{NP}$-hardness. Our reduction further implies that, unless $\mathcal{P} = \mathcal{NP}$, there exist no polynomial-time approximation algorithms with approximation ratio smaller than $3/2$ for these problems.

On extension-parallel temporal graphs, however, we obtain polynomial-time algorithms for the shortest duration path interdiction problem and the shortest traversal path interdiction problem. This result can be transferred to the static shortest path interdiction problem, where it also represents a new result.

## 2 Problem Definition

A directed (discrete-time) *temporal graph* $G$ consists of a nonempty, finite set $V$ of nodes and a finite set $R$ of *temporal arcs*. As usual, we denote the number of nodes and the number of (temporal) arcs in the graph by $n$ and $m$, respectively. A temporal arc $r \in R$ has four attributes, namely its *start node* $\alpha(r) \in V$, its *end node* $\omega(r) \in V$, its *start time* $\tau(r) \in \mathbb{Q}$, and its *traversal time* $\lambda(r) \in \mathbb{Q}_{\geq 0}$. When traversing a temporal arc $r \in R$, the *arrival time* of $r$ is $\tau(r) + \lambda(r)$. A *temporal path* $P = (r_1, \dots, r_k)$ is a sequence of temporal arcs such that, for each $i \in \{1, \dots, k-1\}$, it holds that $\omega(r_i) = \alpha(r_{i+1})$ and $\tau(r_i) + \lambda(r_i) \leq \tau(r_{i+1})$, i.e., the end node of each arc is the start node of the next arc in the path and the arrival time of each arc is less than or equal to the start time of the next arc.[2] For two nodes $s, t \in V$, a temporal path $P = (r_1, \dots, r_k)$ is called a *(temporal) $s$-$t$-path* if $\alpha(r_1) = s$ and $\omega(r_k) = t$. Given a temporal graph $G = (V, R)$, the *underlying static graph* $G^{\text{stat}} = (V^{\text{stat}}, R^{\text{stat}})$ is the (directed) static graph with the same nodes and arcs obtained by disregarding the start times and traversal times of the arcs. A temporal graph is called *acyclic* if its underlying static graph is acyclic, i.e., its underlying static graph does not contain any directed cycle.

While the notion of a "shortest" $s$-$t$-path is straightforward in static graphs, temporal graphs allow for various interpretations of the term "shortest". In this paper, we study the four quality measures for $s$-$t$-paths that are used in [38].

▶ **Definition 1.** *Let $G$ be a temporal graph, $s \neq t$ two nodes in $G$, and $P = (r_1, \dots, r_k)$ a temporal $s$-$t$-path.*

- *The* start time *of $P$ is defined as* $\text{start}(P) \coloneqq \tau(r_1)$.
- *The* arrival time *of $P$ is defined as* $\text{arriv}(P) \coloneqq \tau(r_k) + \lambda(r_k)$.
- *The* duration *of $P$ is defined as* $\text{dura}(P) \coloneqq \text{arriv}(P) - \text{start}(P)$.
- *The* traversal time *of $P$ is defined as* $\text{trav}(P) \coloneqq \sum_{i=1}^{k} \lambda(r_i)$.

▶ **Definition 2.** *Let $G$ be a temporal graph and $s \neq t$ two nodes in $G$.*

- *A* latest start path *is an $s$-$t$-path with maximum start time. The* latest start time *in $G$, denoted by* $\text{LS}(G)$*, is defined as the start time of a latest start path in $G$.*
- *An* earliest arrival path *is an $s$-$t$-path with minimum arrival time. The* earliest arrival time *in $G$, denoted by* $\text{EA}(G)$*, is defined as the arrival time of an earliest arrival path in $G$.*

[2] Note that this definition allows a path to visit the same node (or even traverse the same arc) several times. Except for some results obtained in the setting with waiting time constraints considered in Section 4.2, however, all our results also hold when restricting to *simple* paths that do not visit any node more than once.

- *A* shortest duration path *is an $s$-$t$-path with minimum duration. The* shortest duration *in $G$, denoted by* $\mathrm{SD}(G)$*, is defined as the duration of a shortest duration path in $G$.*
- *A* shortest traversal path *is an $s$-$t$-path with minimum traversal time. The* shortest traversal time *in $G$, denoted by* $\mathrm{ST}(G)$*, is defined as the traversal time of a shortest traversal path in $G$.*

*If no $s$-$t$-path exists in $G$,* $\mathrm{LS}(G)$ *is set to* $-\infty$*, whereas* $\mathrm{EA}(G)$*,* $\mathrm{SD}(G)$*, and* $\mathrm{ST}(G)$ *are set to* $+\infty$*.*

Note that earliest arrival paths are called foremost paths and latest start paths are called reverse-foremost paths in [38]. Next, the four versions of the temporal shortest path interdiction problem are defined.

▶ **Definition 3.** *For an objective* $\mathrm{OBJ} \in \{\mathrm{LS}, \mathrm{EA}, \mathrm{SD}, \mathrm{ST}\}$*, the temporal* OBJ *path interdiction problem (T-*OBJ*P-IP) is defined as follows.*

| | |
|---|---|
| *INSTANCE:* | *A temporal graph $G = (V, R)$, two nodes $s \neq t$ in $G$, a budget $B \in \mathbb{Q}_{>0}$, and removal costs $c : R \to \mathbb{Q}_{\geq 0}$* |
| *TASK:* | *Find a subset $S \subseteq R$ of arcs with $\sum_{r \in S} c(r) \leq B$ such that $\mathrm{OBJ}(G_S)$ is maximized (minimized in the case that $\mathrm{OBJ} = \mathrm{LS}$), where $G_S \coloneqq (V, R \setminus S)$.* |

*A solution $S \subseteq R$ of T-*OBJ*P-IP with $\sum_{r \in S} c(r) \leq B$ is called an* interdiction strategy *and the arcs in $S$ are called* interdicted*. Further, if no temporal path from a node $u$ to another node $v$ exists after the arcs in $S$ have been removed, we say that the interdiction strategy $S$* separates *$u$ from $v$ or that the pair $(u, v)$ is* separated *by $S$.*

## 3 Polynomial-Time Algorithms and Complexity Results

In this section, we analyze the complexity of each of the four introduced versions of temporal shortest path interdiction. It is shown that two versions can be solved in polynomial time and the other two versions are strongly $\mathcal{NP}$-hard. On extension-parallel temporal graphs, however, the two hard versions are shown to be solvable in polynomial time.

### 3.1 Temporal Latest Start Path Interdiction

In this section, we present a polynomial-time algorithm to solve T-LSP-IP. This is a surprising result as the static shortest path interdiction problem is known to be strongly $\mathcal{NP}$-hard [3]. The main reason for the polynomial-time solvability of T-LSP-IP is that the obtained objective value only depends on the first arc that is used by a latest start path in the interdicted graph $G_S$.

In this section, we let $\tau_1 < \tau_2 < \cdots < \tau_l$ denote the distinct start times of outgoing arcs of $s$ in $G$ sorted in increasing order. Further, for $k \in \{1, \ldots, l\}$, we define $G^{\mathrm{LS},k}$ as the temporal graph that results from $G$ by removing all outgoing arcs of $s$ with start time at most $\tau_k$. For completeness, we also define $G^{\mathrm{LS},0} \coloneqq G$. Our algorithm is based on the following proposition.

▶ **Proposition 4.** *Let $k \in \{1, \ldots, l\}$. There exists an interdiction strategy $S^k$ that separates $s$ from $t$ in $G^{\mathrm{LS},k}$ if and only if there exists an interdiction strategy $S$ in $G$ with objective value at most $\tau_k$.*

**Proof.** Let $S^k$ be an interdiction strategy that separates $s$ from $t$ in $G^{\mathrm{LS},k}$. Then, after interdicting the same set $S := S^k$ of arcs in $G$, no $s$-$t$-path in $G_S$ can start with an arc with start time strictly larger than $\tau_k$ (otherwise, the path would also be an $s$-$t$-path in $G^{\mathrm{LS},k}_{S^k}$). Hence, all $s$-$t$-paths in $G_S$ have start time at most $\tau_k$, i.e., $S$ has objective value at most $\tau_k$.

Conversely, let $S$ be an interdiction strategy in $G$ with objective value at most $\tau_k$. Then, no $s$-$t$-path in $G_S$ can have start time strictly larger than $\tau_k$, so the interdiction strategy $S^k := S \cap R^{\mathrm{LS},k}$, where $R^{\mathrm{LS},k}$ is the arc set of $G^{\mathrm{LS},k}$, separates $s$ from $t$ in $G^{\mathrm{LS},k}$. ◀

The idea of the algorithm is to use binary search in order to find $k^\star \in \{1, \dots, l\}$ such that $s$ can be separated from $t$ in $G^{\mathrm{LS},k^\star}$, but $s$ cannot be separated from $t$ in $G^{\mathrm{LS},k^\star-1}$. Such a $k^\star$ exists whenever $s$ cannot already be separated from $t$ in the whole graph $G = G^{\mathrm{LS},0}$, i.e., whenever the optimal objective value is not equal to $-\infty$. Consequently, in order to obtain a polynomial-time algorithm for T-LSP-IP, it only remains to show that deciding whether a node $s$ can be separated from another node $t$ with a given interdiction budget in an arbitrary temporal graph is possible in polynomial time.

In a static graph, this question can be answered easily by computing a minimum $s$-$t$-cut and comparing its cost to the given interdiction budget $B$. Hence, we now describe how the question in an arbitrary temporal graph $H = (V, R)$ can be reduced to the static case. To this end, we use a graph construction that is similar to [38] and to the construction of time-expanded networks in the context of dynamic flows [32]. The constructed graph is therefore called the *time-expanded graph* of $H$ and denoted by $H^{\mathrm{te}} = (V^{\mathrm{te}}, R^{\mathrm{te}})$. We start by defining the set of *crucial times* by $T := \cup_{r \in R}\{\tau(r), \tau(r) + \lambda(r)\}$. For easier notation, we write $T = \{\phi_1, \dots, \phi_j\}$, where the crucial times are indexed in increasing order. For each $v \in V$ and $\phi \in T$, there exists a node $(v, \phi)$ in $V^{\mathrm{te}}$. For each $i \in \{1, \dots, j-1\}$ and for each $v \in V$, there exists an arc from $(v, \phi_i)$ to $(v, \phi_{i+1})$ with removal cost $B + 1$ (i.e., it cannot be interdicted). This arc represents waiting at node $v$ of the temporal graph until the next crucial time. Further, for each arc $r \in R$, there exists an arc in $R^{\mathrm{te}}$ from $(\alpha(r), \tau(r))$ to $(\omega(r), \tau(r) + \lambda(r))$ with removal cost $c(r)$, which represents traversing arc $r$ in the temporal graph. We define $s^{\mathrm{te}} := (s, \phi_1)$ and $t^{\mathrm{te}} := (t, \phi_j)$. If the temporal graph $H$ has $n$ nodes and $m$ arcs, its time-expanded graph has $n \cdot |T| \in \mathcal{O}(n \cdot m)$ nodes and $n \cdot (|T| - 1) + m \in \mathcal{O}(n \cdot m)$ arcs. Hence, the size of the time-expanded graph is polynomial in the size of the temporal graph (in contrast to time-expanded networks used in the context of dynamic flows).
The following observation follows directly from the construction of $H^{\mathrm{te}}$.

▶ **Observation 5.** *There exists an interdiction strategy separating $s$ from $t$ in $H$ if and only if there exists an interdiction strategy separating $s^{\mathrm{te}}$ from $t^{\mathrm{te}}$ in $H^{\mathrm{te}}$.*

Applying the previously described algorithm together with Proposition 4 and Observation 5 yields the main theorem of this section.

▶ **Theorem 6.** *There exists a polynomial-time algorithm for T-LSP-IP with running time in $\mathcal{O}(\log(l) \cdot T_{\mathrm{MC}}(n \cdot m, n \cdot m))$, where $l \leq m$ is the number of distinct start times of outgoing arcs of $s$ and $T_{\mathrm{MC}}(n \cdot m, n \cdot m)$ is the time required to compute a minimum $s$-$t$-cut in a static graph with $n \cdot m$ nodes and $n \cdot m$ arcs.*

## 3.2 Temporal Earliest Arrival Path Interdiction

In this section, we present a polynomial-time algorithm to solve T-EAP-IP. Similar to T-LSP-IP, the reason for the problem's polynomial-time solvability is that the obtained objective value only depends on the last arc that is used by an earliest arrival path in the interdicted

graph $G_S$. Indeed, an instance of T-EAP-IP can be transformed into an equivalent instance of T-LSP-IP by inverting the direction of all arcs and adjusting the start times and traversal times appropriately. This is described in the following.

Let $G = (V, R)$ be the temporal graph in an instance of T-EAP-IP. We construct a graph $G^{\mathrm{LS}} = (V, R^{\mathrm{LS}})$ for an instance of T-LSP-IP. The *maximum arrival time* in $G$ is defined as $\Phi := \max_{r \in R} \tau(r) + \lambda(r)$. For each $r \in R$, an arc $r'$ is added to $R^{\mathrm{LS}}$ with $\alpha(r') := \omega(r)$, $\omega(r') := \alpha(r)$, $\tau(r') := \Phi - \tau(r) - \lambda(r)$, and $\lambda(r') := \lambda(r)$. The arcs $r$ and $r'$ are called *associated*. Further, an interdiction strategy $S$ in $G$ and the interdiction strategy $S'$ in $G^{\mathrm{LS}}$ consisting of the arcs in $G^{\mathrm{LS}}$ that are associated with those in $S$ are also called *associated*. Defining $s^{\mathrm{LS}} := t$ and $t^{\mathrm{LS}} := s$, $B^{\mathrm{LS}} := B$, and $c^{\mathrm{LS}}(r') := c(r)$ for each pair of associated arcs $r$ and $r'$, it is then easy to see that mapping an interdiction strategy $S$ in $G$ to its associated interdiction strategy $S'$ in $G^{\mathrm{LS}}$ defines a bijection between the sets of interdiction strategies in the two graphs. In the following, the instance of T-EAP-IP is denoted by $(G, s, t)$ and the constructed instance of T-LSP-IP by $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$. We can then show the following one-to-one correspondence between temporal paths in $G$ and $G^{\mathrm{LS}}$.

▶ **Proposition 7.** *Let $r_i$ and $r'_i$ be associated arcs for each $i \in \{1, \dots, k\}$, and let $S$ and $S'$ be associated interdiction strategies in $G$ and $G^{\mathrm{LS}}$, respectively. Then $P' = (r'_1, \dots, r'_k)$ is a temporal $s^{\mathrm{LS}}$-$t^{\mathrm{LS}}$-path in $G^{\mathrm{LS}}_{S'}$ if and only if $P = (r_k, \dots, r_1)$ is a temporal $s$-$t$-path in $G_S$.*

**Proof.** If $P' = (r'_1, \dots, r'_k)$ is a temporal $s^{\mathrm{LS}}$-$t^{\mathrm{LS}}$-path in $G^{\mathrm{LS}}_{S'}$, then $r'_i \notin S'$ for $i = 1, \dots, k$. Hence, since $S'$ and $S$ are associated, we obtain that $r_i \notin S$ for $i = 1, \dots, k$. Moreover, $t = s^{\mathrm{LS}} = \alpha(r'_1) = \omega(r_1)$, $s = t^{\mathrm{LS}} = \omega(r'_k) = \alpha(r_k)$, and for each $i \in \{1, \dots, k-1\}$, we have $\alpha(r_i) = \omega(r'_i) = \alpha(r'_{i+1}) = \omega(r_{i+1})$ and

$$\tau(r_{i+1}) + \lambda(r_{i+1}) = \Phi - \tau(r'_{i+1}) \leq \Phi - \tau(r'_i) - \lambda(r'_i) = \Phi - \tau(r'_i) - \lambda(r_i) = \tau(r_i).$$

Thus, $P = (r_k, \dots, r_1)$ is a temporal $s$-$t$-path in $G_S$ as claimed. The inverse direction can be shown along the same lines. ◀

We call paths $P$ and $P'$ as in Proposition 7 *associated* in the following. Proposition 7 allows us to show the following relationship between the objective values of associated interdiction strategies for $(G, s, t)$ and $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$.

▶ **Corollary 8.** *An interdiction strategy $S$ for $(G, s, t)$ has objective value $v$ if and only if the associated interdiction strategy $S'$ for $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$ has objective value $\Phi - v$.*

**Proof.** Given an interdiction strategy $S$ with objective value $v$ and its associated interdiction strategy $S'$, let $P$ be an earliest arrival path in $G_S$. Then, $P$ has arrival time $v$ and by Proposition 7, the associated path $P'$ is a temporal path in $G^{\mathrm{LS}}_{S'}$, whose start time is $\Phi - v$. For the sake of a contradiction, suppose that there exists a path $\bar{P}'$ in $G^{\mathrm{LS}}_{S'}$ with start time $\Phi - \bar{v} > \Phi - v$. Then, by Proposition 7, the path $\bar{P}$ that is associated to $\bar{P}'$ is a temporal path in $G_S$ and its arrival time is $\bar{v} < v$, which is a contradiction to $P$ being an earliest arrival path in $G_S$. Hence, the interdiction strategy $S'$ for $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$ has objective value $\Phi - v$. The inverse direction can be shown along the same lines. ◀

Corollary 8 immediately yields the following result.

▶ **Corollary 9.** *An interdiction strategy $S$ is optimal for $(G, s, t)$ if and only if its associated interdiction strategy $S'$ is optimal for $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$.*

Corollary 9 and the algorithm presented in Section 3.1 yield the main result of this section.

▶ **Theorem 10.** *There exists a polynomial-time algorithm for T-EAP-IP with running time in* $\mathcal{O}(\log(l) \cdot T_{\mathrm{MC}}(n \cdot m, n \cdot m))$*, where* $l \leq m$ *is the number of distinct arrival times of incoming arcs of* $t$ *and* $T_{\mathrm{MC}}(n \cdot m, n \cdot m)$ *is the time required to compute a minimum* $s$*-*$t$*-cut in a static graph with* $n \cdot m$ *nodes and* $n \cdot m$ *arcs.*

## 3.3 Temporal Shortest Duration Path Interdiction and Temporal Shortest Traversal Path Interdiction

In this section, we show that T-SDP-IP and T-STP-IP are strongly $\mathcal{NP}$-hard, even for unit removal costs and if the underlying static graph is acyclic. Moreover, the reduction implies an inapproximability result. We also show, however, that both problems are solvable in polynomial time if the graph is extension-parallel. This result is also shown for the static problem S-SP-IP.

The proof of strong $\mathcal{NP}$-hardness is similar to the proof in [6], where it is shown that finding a multicut in directed acyclic graphs is $\mathcal{APX}$-hard. The reduction is performed from the strongly $\mathcal{NP}$-hard MAX2SAT problem, which is defined as follows.

| | |
|---|---|
| INSTANCE: | A set $X = \{x_1, \ldots, x_\zeta\}$ of boolean variables, a set $C = \{c_1, \ldots, c_\mu\}$ of clauses each containing two literals, and a positive integer $\delta < \mu$ |
| QUESTION: | Is there a truth assignment for the variables that satisfies at least $\delta$ clauses? |

Given an instance of MAX2SAT, we construct a temporal graph with removal costs and a corresponding budget. This graph has the property that no $s$-$t$-path waits in any node except for $s$ and $t$, which means that, for each feasible interdiction strategy, the objective values in T-SDP-IP and T-STP-IP are identical. Hence, the resulting instances of T-SDP-IP and T-STP-IP are equivalent in this case. Thus, we present the construction and the corresponding proofs only for T-SDP-IP in the following. An example for the construction is provided in Figure 1.

Unless explicitly stated otherwise, all arcs within this construction have start time 0, traversal time 0, and removal cost $B + 1$ (i.e., they cannot be interdicted). We show later that only a slight modification of the construction is necessary in the case of unit removal costs. For each variable $x_i \in X$, there is a *variable gadget* consisting of a directed path with trace $(u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4})$ where only the arcs from $u_{i,1}$ to $u_{i,2}$ and from $u_{i,3}$ to $u_{i,4}$ can be interdicted at a removal cost of $N \coloneqq \mu + 1$. Interdicting the arc from $u_{i,1}$ to $u_{i,2}$ is later identified with setting $x_i$ to true and interdicting the arc from $u_{i,3}$ to $u_{i,4}$ is identified with setting $x_i$ to false. For each clause $c_j \in C$, there is a *clause gadget* consisting of a directed path with trace $(v_{j,1}, v_{j,2}, v_{j,3}, v_{j,4})$ where only the arcs from $v_{j,1}$ to $v_{j,2}$ and from $v_{j,3}$ to $v_{j,4}$ can be interdicted at a removal cost of one.

We next describe the arcs that connect the variable gadgets to the clause gadgets. For a clause $c_j = \hat{x}_i \vee \hat{x}_k$, where $\hat{x}_i \in \{x_i, \overline{x}_i\}$ and $\hat{x}_k \in \{x_k, \overline{x}_k\}$, we call $\hat{x}_i$ the *first literal* and $\hat{x}_k$ the *second literal* of clause $c_j$. For each clause, arcs are then added as follows depending on its first and second literal: If the first literal of clause $c_j$ is $x_i$ ($\overline{x}_i$), there exists an arc from $u_{i,2}$ to $v_{j,1}$ (from $u_{i,4}$ to $v_{j,1}$). If the second literal of clause $c_j$ is $x_k$ ($\overline{x}_k$), there exists an arc from $u_{k,2}$ to $v_{j,3}$ (from $u_{k,4}$ to $v_{j,3}$).

The construction is continued by adding another six nodes $s_1, s_2, s_3, t_1, t_2$, and $t_3$ to the graph. For each $i \in \{1, \ldots, \zeta\}$, there exists an arc from $s_1$ to $u_{i,1}$ and an arc from $u_{i,4}$ to $t_1$. For each $i \in \{1, \ldots, \zeta\}$, there exists an arc from $s_2$ to $u_{i,1}$ and another arc from $s_2$ to $u_{i,3}$. Further, for each $j \in \{1, \ldots, \mu\}$, there exist arcs from $v_{j,2}$ to $t_2$ and from $v_{j,4}$ to $t_2$. Finally, for each $j \in \{1, \ldots, \mu\}$, there exists an arc from $s_3$ to $v_{j,1}$ and an arc from $v_{j,4}$ to $t_3$.

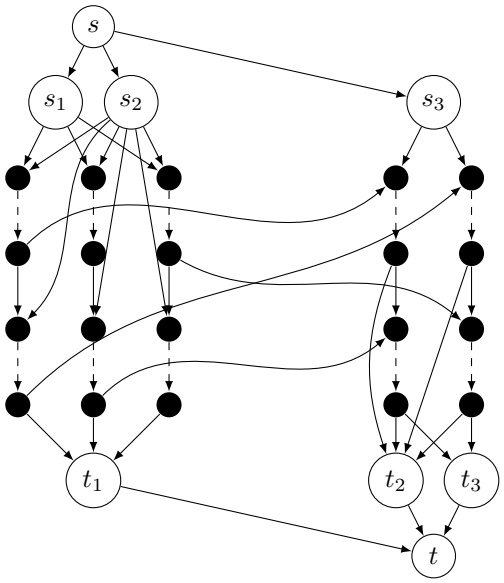



**Figure 1** The constructed graph for $X = \{x_1, x_2, x_3\}$ and $C = \{x_1 \vee \overline{x}_2, \overline{x}_1 \vee x_3\}$. The three variable gadgets for $x_1$, $x_2$, and $x_3$ from left to right are shown on the left and the clause gadgets for $c_1$ and $c_2$ from left to right are shown on the right. Only the dashed arcs can be interdicted.

We finish the construction by adding the nodes $s$ and $t$ to the graph. For each $k \in \{1, 2, 3\}$, there exists an arc from $s$ to $s_k$ with start time $1 - k$ and traversal time $k - 1$, and an arc from $t_k$ to $t$ with traversal time $3 - k$ (but start time 0).

The budget is chosen to be $B \coloneqq N \cdot \zeta + 2 \cdot \mu - \delta$, which completes the construction of the problem instance.

We show strong $\mathcal{NP}$-hardness by proving that there exists a truth assignment for the variables that satisfies at least $\delta$ clauses in the instance of MAX2SAT if and only if there exists a solution for the constructed instance with objective value at least 3. To this end, the following auxiliary result is required.

▶ **Lemma 11.** *Let $S$ be a solution of the constructed instance. The objective value of $S$ is larger than or equal to 3 if and only if the pairs $(s_1, t_1), (s_2, t_2)$, and $(s_3, t_3)$ are separated by $S$.*

**Proof.** If one of the pairs $(s_1, t_1), (s_2, t_2)$, or $(s_3, t_3)$ is not separated by $S$, it follows immediately that, after interdiction, there exists an $s$-$t$-path with duration 2. Hence, the solution $S$ has objective value at most 2. To show the other direction, assume that the objective value of the solution is strictly less than 3 and let $P_{\mathrm{SD}}$ be a shortest duration path in $G_S$. If $P_{\mathrm{SD}}$ visits both $s_k$ and $t_k$ for some $k \in \{1, 2, 3\}$, we are done. If this is not the case, there are three possible pairs of nodes, one of which must be visited by $P_{\mathrm{SD}}$ since its duration is strictly less than 3 and every temporal $s$-$t$-path in $G_S$ must visit one of the $s_k$ and one of the $t_k$.

**Case 1:** $P_{\mathrm{SD}}$ visits $s_1$ and $t_2$

This means there exists a subpath $P$ of $P_{\mathrm{SD}}$ from $s_1$ to $t_2$. The first arc in $P$ leads into one of the nodes $u_{i,1}$ for some $i \in \{1, \ldots, \zeta\}$. By replacing the first arc in $P$ with the arc starting in $s_2$ and ending in $u_{i,1}$, we obtain a path from $s_2$ to $t_2$.

**Case 2:** $P_{\mathrm{SD}}$ visits $s_2$ and $t_3$

This means there exists a subpath $P$ of $P_{\mathrm{SD}}$ from $s_2$ to $t_3$. The last arc in $P$ starts from one of the nodes $v_{j,4}$ for some $j \in \{1, \ldots, \mu\}$. By replacing the last arc in $P$ with the arc starting in $v_{j,4}$ and ending in $t_2$, we again obtain a path from $s_2$ to $t_2$.

**Case 3:** $P_{\mathrm{SD}}$ visits $s_1$ and $t_3$

The first and the last arc in the subpath of $P_{\mathrm{SD}}$ from $s_1$ to $t_3$ can be replaced as in the previous two cases, which again yields a path from $s_2$ to $t_2$. ◀

Lemma 11 allows proving strong $\mathcal{NP}$-hardness of T-SDP-IP and T-STP-IP.

▶ **Theorem 12.** *T-SDP-IP and T-STP-IP are strongly $\mathcal{NP}$-hard even on acyclic graphs.*

**Proof.** We show that there exists a truth assignment for the variables that satisfies at least $\delta$ clauses in the instance of MAX2SAT if and only if there exists a solution for the constructed T-SDP-IP instance with objective value at least 3.

First, let $x$ be a truth assignment that satisfies at least $\delta$ clauses. We construct an interdiction strategy for the instance of T-SDP-IP with objective value at least 3 as follows. For each $i \in \{1, \dots, \zeta\}$, we interdict the arc from $u_{i,1}$ to $u_{i,2}$ if $x_i$ is true and the arc from $u_{i,3}$ to $u_{i,4}$ if $x_i$ is false. For each $j \in \{1, \dots, \mu\}$, we interdict the arc from $v_{j,1}$ to $v_{j,2}$ if the second literal in clause $c_j$ is fulfilled, the arc from $v_{j,3}$ to $v_{j,4}$ if the second literal of $c_j$ is not fulfilled, but the first is, and both of these arcs if none of the literals are fulfilled. This yields an interdiction strategy $S$ that interdicts $\zeta$ arcs of cost $N$ and at most $2 \cdot \mu - \delta$ arcs of cost 1 and, hence, does not exceed the budget.

Due to Lemma 11, it remains to show that the pair $(s_k, t_k)$ is separated by $S$ for each $k \in \{1, 2, 3\}$. Any path from $s_1$ to $t_1$ has trace $(s_1, u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4}, t_1)$ for some $i \in \{1, \dots, \zeta\}$. As either the arc from $u_{i,1}$ to $u_{i,2}$ or the arc from $u_{i,3}$ to $u_{i,4}$ is interdicted, the pair $(s_1, t_1)$ is separated by $S$. Moreover, the analogous argument applied to the clause gadgets shows that the pair $(s_3, t_3)$ is separated by $S$.

To show that the pair $(s_2, t_2)$ is separated by $S$, note that each path from $s_2$ to $t_2$ contains a subpath with trace $(u_{i,a}, u_{i,a+1}, v_{j,b}, v_{j,b+1})$ where $i \in \{1, \dots, \zeta\}, j \in \{1, \dots, \mu\}$, and $a, b \in \{1, 3\}$. We interdict either the arc from $u_{i,a}$ to $u_{i,a+1}$ if the $(^{b+1}/_2)$-th literal of clause $c_j$ is fulfilled or the arc from $v_{j,b}$ to $v_{j,b+1}$ if it is not. Hence, the pair $(s_2, t_2)$ is separated by $S$ and the objective value of $S$ is at least 3 due to Lemma 11.

For the inverse direction, let $S \subseteq R$ be a removal strategy with $c(S) \leq B = N \cdot \zeta + 2 \cdot \mu - \delta$ and objective value at least 3. In particular, this means that, for each $k \in \{1, 2, 3\}$, the pair $(s_k, t_k)$ is separated by $S$ due to Lemma 11.

In order to separate the pair $(s_1, t_1)$, one arc has to be removed per variable gadget. If more than one arc is removed in any variable gadget, the total removal cost of interdicted arcs in the variable gadgets is at least $N \cdot \zeta + N = N \cdot \zeta + \mu + 1$, which leaves only a budget of $\mu - \delta - 1 < \mu$ for interdicting arcs in the clause gadgets. Hence, there must exist at least one clause gadget in which none of the arcs are interdicted. This implies that the pair $(s_3, t_3)$ is not separated by $S$, which is a contradiction. Overall, this means that, for each $i \in \{1, \dots, \zeta\}$, either the arc from $u_{i,1}$ to $u_{i,2}$ is interdicted, in which case we set $x_i$ to true, or the arc from $u_{i,3}$ to $u_{i,4}$ is interdicted, in which case we set $x_i$ to false.

It remains to show that the resulting truth assignment fulfills at least $\delta$ clauses. As interdicting one arc per variable gadget already costs $N \cdot \zeta$, there is a budget of $2 \cdot \mu - \delta$ left for interdicting arcs in the clause gadgets. In order to separate the pair $(s_3, t_3)$, at least one of the two removable arcs must be removed in each clause gadget. Hence, there are at least $\delta$ clause gadgets in which only one of the arcs is removed. We finish the proof by showing that $x$ fulfills all the corresponding clauses $c_j$.

To this end, we first assume that the arc from $v_{j,3}$ to $v_{j,4}$ is interdicted. If the first literal in $c_j$ is $x_i$, then there exists a path in $G$ with trace $(s_2, u_{i,1}, u_{i,2}, v_{j,1}, v_{j,2}, t_2)$. As the arc from $v_{j,1}$ to $v_{j,2}$ is not interdicted and the pair $(s_2, t_2)$ must be separated by $S$, this means that the arc from $u_{i,1}$ to $u_{i,2}$ must be interdicted and, hence, that $x_i$ is set to true, which

shows that $x$ fulfills $c_j$. If the first literal in $c_j$ is $\overline{x}_i$, then the same arguments hold for the path in $G$ with trace $(s_2, u_{i,3}, u_{i,4}, v_{j,1}, v_{j,2}, t_2)$. The proof for the case when the arc from $v_{j,1}$ to $v_{j,2}$ is interdicted is along the same lines. Hence, at least $\delta$ clauses are fulfilled by $x$, which completes the proof. ◀

Since any solution of the constructed T-SDP-IP instance that does *not* have objective value at least 3 has objective value at most 2, the proof of Theorem 12 further implies the following inapproximability result.

▶ **Corollary 13.** *Unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial-time approximation algorithm with approximation ratio smaller than $3/2$ for T-SDP-IP or T-STP-IP, even on acyclic graphs.*

In the case of T-SDP-IP, the constructed instance in the reduction can easily be adjusted such that all traversal times are zero. To do so, all traversal times of the outgoing arcs of $s$ are set to 0 and, for each incoming arc of $t$, the start time is increased by its traversal time and the traversal time is then set to 0. Hence, the results on T-SDP-IP from Theorem 12 and Corollary 13 are also valid for the definition of temporal graphs used in [27].

In the case of T-STP-IP, however, using nonzero traversal times within the reduction is necessary. Indeed, the results on T-STP-IP from Theorem 12 and Corollary 13 do *not* hold for the definition in [27] (unless $\mathcal{P} = \mathcal{NP}$) since T-STP-IP is solvable in polynomial time if all traversal times are zero as it then reduces to the question whether $s$ can be separated from $t$ by an interdiction strategy. It is, however, questionable, whether T-STP-IP has a meaningful interpretation in this case.

We continue by showing that the results of Theorem 12 and Corollary 13 (with a slight modification of the approximation ratio) also hold for instances with unit removal costs and strictly positive traversal times.

The restriction to unit removal costs can be achieved by replacing each arc $r$ in the constructed graph by $c(r)$ identical copies with unit removal cost. Any interdiction strategy can then be assumed to either remove all of these identical copies or none of them. Moreover, since all removal costs have been polynomial in the numbers of variables and clauses of the given MAX2SAT instance, the constructed instance with unit removal costs is still of polynomial size, so the arguments in the proof carry over to this instance.

For the restriction to strictly positive traversal times, note that the constructed graph $G$ is acyclic. In particular, the graph $G - \{s, t\}$ is acyclic. Let $\sigma : V \to \{1, \dots, n-2\}$ be a topological sorting of the nodes in $G - \{s, t\}$, i.e., for each arc $r$, it holds that $\sigma(\alpha(r)) < \sigma(\omega(r))$. This topological sorting is used to slightly modify the start and traversal times of the arcs in $G - \{s, t\}$. Formally, a function $\bar{\sigma} : V \to \{1, \dots, n-2\}$ is constructed from the topological sorting by setting $\bar{\sigma}(v) \coloneqq \sigma(v)$ if $v \notin \{s_1, s_2, s_3, t_1, t_2, t_3\}$, and $\bar{\sigma}(s_i) \coloneqq 1$ and $\bar{\sigma}(t_i) \coloneqq n-2$ for each $i \in \{1, 2, 3\}$. We then redefine the start and traversal times in $G$. To this end, let $\varepsilon \in (0, 1)$. For each arc $r$ that is not incident to $s$ or $t$, we set the start time to $-\varepsilon/2 \cdot (n-3) \cdot (n - 2 - \bar{\sigma}(\alpha(r)))$ and the traversal time to $\varepsilon/2 \cdot (n-3) \cdot (\bar{\sigma}(\omega(r)) - \bar{\sigma}(\alpha(r)))$, which means that it arrives in $\omega(r)$ at time $-\varepsilon/2 \cdot (n-3) \cdot (n - 2 - \bar{\sigma}(\omega(r)))$. We further set the start time of the arc from $s$ to $s_1$ to $-\varepsilon$ and its traversal time to $\varepsilon/2$, and we decrease the traversal times of the other two outgoing arcs of $s$ by $\varepsilon/2$. Hence, all outgoing arcs of $s$ have arrival time $-\varepsilon/2$. Moreover, we set the traversal time of the arc from $t_3$ to $t$ to $\varepsilon$.

The proof of Theorem 12 for this new instance is along the same lines as before and the statement of Corollary 13 must be slightly changed (see Corollary 14). Note that the graph with the updated start and traversal times does not admit waiting in any node except for $s$ and $t$ as, for any node $v \in V \setminus \{s, t\}$, all incoming arcs arrive and all outgoing arcs start at time $-\varepsilon/2 \cdot (n-3) \cdot (n - 2 - \bar{\sigma}(v))$. The result, hence, holds for both problems T-SDP-IP and T-STP-IP.

Hence, when strictly positive traversal times on all arcs are additionally assumed, Theorem 12 still holds, but Corollary 13 has to be adapted as follows:

▶ **Corollary 14.** *Unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial-time approximation algorithm with approximation ratio smaller than $(3/2+\varepsilon)$ for T-SDP-IP and T-STP-IP on acyclic graphs with positive traversal times for any $\varepsilon > 0$.*

### 3.3.1 Polynomial-Time Solvability on Extension-Parallel Graphs

In this section, we show that T-SDP-IP, T-STP-IP, and the static version S-SP-IP are polynomial-time solvable on extension-parallel (temporal) graphs.

A temporal graph consisting of two nodes $s$ and $t$, and a single temporal arc from $s$ to $t$ is called a *temporal one-arc graph*. A temporal graph with two distinguished vertices $s$ (the source) and $t$ (the sink) is *series-parallel* if it is obtained from a set of temporal one-arc graphs by a finite sequence of series compositions (identifying the sink of the first graph with the source of the second graph) and parallel compositions (identifying the sources of the two graphs and identifying the sinks of the two graphs). If, further, for every series composition, one of the two composed graphs is a temporal one-arc graph, the graph is called *extension-parallel*. The definitions of series- and extension-parallel *static* graphs is completely analogous and can be found, e.g., in [19].

The *decomposition tree* $T_G$ of a series-parallel (temporal) graph $G$ is a binary tree, where the leaves represent the arcs in the graph and the inner nodes labeled by S (series composition) or P (parallel composition) represent the types of compositions used to construct the graph. The decomposition tree can be computed in linear time [36] and it can easily be seen that a series-parallel (temporal) graph is extension-parallel if and only if every inner node of $T_G$ that is labeled by S has one child that is a leaf of $T_G$.
The following property of extension-parallel static graphs is used in our algorithm.

▶ **Lemma 15.** *Let $G = (V, R)$ be an extension-parallel static graph. Then there exists a subset $\bar{R} \subseteq R$ of arcs such that*
1. *each $s$-$t$-path in $G$ contains exactly one arc from $\bar{R}$, and*
2. *each arc $r \in \bar{R}$ is contained in exactly one $s$-$t$-path $P_r$ in $G$.*

**Proof.** We present an algorithm that constructs $\bar{R}$ and a corresponding $s$-$t$-path $P_r$ for each $r \in \bar{R}$. Note that, since the graph is acyclic, the path $P_r$ is uniquely determined by the set of arcs it traverses. Hence, we slightly abuse notation and identify each path $P_r$ with the corresponding set of arcs. The idea of the algorithm is to process the nodes in the decomposition tree starting at the leaves by iteratively joining two already processed components of the graph until we reach the root node and obtain the final set $\bar{R}$.

Initially, we set $\bar{R} := R$ and $P_r := \{r\}$ for each $r \in R$ and mark all leaf nodes in the decomposition tree as processed. While not all nodes in the decomposition tree are marked as processed, we take an unprocessed (inner) node $v$ in the decomposition tree whose two children have both been processed. If $v$ is labeled by P, we simply mark $v$ as processed while changing neither the set $\bar{R}$ nor the paths $P_r$, $r \in \bar{R}$. If $v$ is labeled by S, at least one of its children must be a leaf corresponding to an arc $r$. If only one of the children is a leaf node, then we remove $r$ from $\bar{R}$ and delete $P_r$. Further, we add $r$ to all paths $P_{r'}$ for which the leaf node that corresponds to $r'$ is a successor of the non-leaf child of $v$ in the decomposition tree. If both children of $v$ are leaves, then the arc that corresponds to its right child is removed from $\bar{R}$ and added to the path $P_{r'}$, where $r'$ is the arc that corresponds to the left child of $v$. We then mark $v$ as processed and proceed.

To show the correctness of the algorithm, note that each node $v$ in the decomposition tree can be associated with the subgraph $G_v$ of $G$ whose arc set consists of those arcs that correspond to leaf nodes in the decomposition tree that are successors of $v$.

We claim that, after each iteration, for each processed node $v$ that either has no parent (i.e, $v$ is the root node) or whose parent is still unprocessed, it holds that $\bar{R}$ restricted to the arc set of $G_v$ fulfills the properties from the lemma for $G_v$.

This is clearly the case when only the leaves have been processed since every subgraph $G_v$ is then a one-arc graph. Now assume that the claim holds at the beginning of an iteration and let $v$ be the node in the decomposition tree that is processed in the iteration. If $v$ is labeled by P, each $s$-$t$-path in $G_v$ is either completely contained in the graph associated with the left child of $v$ in the decomposition tree or in the graph associated with the right child. Hence, since $\bar{R}$ and the paths $P_r$, $r \in \bar{R}$, are left unchanged, the claim also holds after processing $v$. If the processed node $v$ is labeled by S and both its children are leaves, the claim clearly remains true. If the processed node $v$ is labeled by S and only one of its children is a leaf, then this series composition corresponds to prepending or appending an additional arc to the graph $G_w$, where $w$ denotes the non-leaf child of $v$. Hence, after the series composition, each path in $G_w$ is extended by the arc $r$ that corresponds to the leaf child of $v$, which is precisely what the algorithm does. Moreover, $r$ is removed from $\bar{R}$ and $P_r$ is deleted, which ensures that uniqueness is preserved in both properties from the lemma. Hence, the claim also holds after the iteration, which completes the proof. ◀

Note that the proof of Lemma 15 is constructive and the set $\bar{R}$ together with the paths $P_r$ for $r \in \bar{R}$ can be obtained in $\mathcal{O}(m^2)$ time. In the following, given an extension-parallel temporal graph, we let $\bar{R}$ denote a subset of arcs that satisfies the properties of Lemma 15 in the underlying static graph. For each arc $r \in \bar{R}$, we remove $r$ from the graph and from $\bar{R}$ if the corresponding $s$-$t$-path $P_r$ in the underlying static graph is not a (temporal) $s$-$t$-path in the temporal graph. Note that this does not destroy any temporal $s$-$t$-paths.

The idea of the polynomial-time algorithm to solve T-SDP-IP, T-STP-IP, and the static version S-SP-IP is similar to the idea of the algorithm for T-LSP-IP from Section 3.1. For ease of notation, the following exposition is restricted to T-SDP-IP. It is discussed later how the arguments can be modified for the other two problems.

Let $\text{dura}_1 < \text{dura}_2 < \cdots < \text{dura}_l$ denote the distinct durations of $s$-$t$-paths in $G$ sorted in increasing order. Further, for $k \in \{1, \dots, l\}$, we define $G^{\text{SD},k}$ as the temporal graph that results from $G$ by removing each arc $r \in \bar{R}$ for which $P_r$ has duration at least $\text{dura}_k$. For completeness, we also define $G^{\text{SD},l+1} := G$. The following proposition and its proof are similar to Proposition 4 and the corresponding proof.

▶ **Proposition 16.** *Let $k \in \{1, \dots, l\}$. There exists an interdiction strategy $S^k$ that separates $s$ from $t$ in $G^{\text{SD},k}$ if and only if there exists an interdiction strategy $S$ in $G$ with objective value at least $\text{dura}_k$.*

**Proof.** Let $S^k$ be an interdiction strategy that separates $s$ from $t$ in $G^{\text{SD},k}$. Then, after interdicting the same set $S := S^k$ of arcs in $G$, no $s$-$t$-path $P$ in $G_S$ can have duration less than $\text{dura}_k$ (otherwise, $P$ would also be an $s$-$t$-path in $G^{\text{SD},k}_{S^k}$ as (1) no arc in $P$ is in $S = S^k$, and (2) the unique arc $r$ in $P$ contained in $\bar{R}$ satisfies $P_r = P$ and, thus, $\text{dura}(P_r) = \text{dura}(P) < \text{dura}_k$). Hence, all $s$-$t$-paths in $G_S$ have duration at least $\text{dura}_k$, i.e., $S$ has objective value at least $\text{dura}_k$.

Conversely, let $S$ be an interdiction strategy in $G$ with objective value at least $\text{dura}_k$. Then, no $s$-$t$-path in $G_S$ can have duration less than $\text{dura}_k$, so the interdiction strategy $S^k := S \setminus \{r \in \bar{R} \,|\, \text{dura}(P_r) \geq \text{dura}_k\}$ separates $s$ from $t$ in $G^{\text{SD},k}$. ◀

As in the algorithm presented in Section 3.1, the idea of the algorithm is to use binary search in order to find $k^\star \in \{1, \dots, l\}$ such that $s$ can be separated from $t$ in $G^{\mathrm{SD},k^\star}$, but $s$ cannot be separated from $t$ in $G^{\mathrm{SD},k^\star+1}$. Such a $k^\star$ exists whenever $s$ cannot already be separated from $t$ in the whole graph $G = G^{\mathrm{SD},l+1}$, i.e., whenever the optimal objective value is not equal to $+\infty$.

As shown in Section 3.1, deciding whether a node $s$ and can be separated from another node $t$ with a given interdiction budget in an arbitrary temporal graph is possible in polynomial time. Further, Lemma 15 implies that the total number of $s$-$t$-paths is bounded by the number of arcs in the graph. Consequently, the number $l$ of distinct durations of $s$-$t$-paths is polynomial in the input size. Altogether, the proposed algorithm runs in polynomial time.

To extend the result to the problems T-STP-IP and S-SP-IP, one observes the distinct traversal times or lengths of $s$-$t$-paths, respectively, to construct the subgraphs used in the algorithm. All arguments then work along the same lines.
The following theorem summarizes the main results of this section.

▶ **Theorem 17.** *There exist polynomial-time algorithms for T-SDP-IP, T-STP-IP, and S-SP-IP on extension-parallel (temporal) graphs with running time in* $\mathcal{O}(m^2 + \log(m) \cdot T_{\mathrm{MC}}(n \cdot m, n \cdot m))$ *for the temporal versions and running time in* $\mathcal{O}(m^2 + \log(m) \cdot T_{\mathrm{MC}}(n, m))$ *for the static version, where* $T_{\mathrm{MC}}(\bar{n}, \bar{m})$ *is the time required to compute a minimum s-t-cut in a static graph with* $\bar{n}$ *nodes and* $\bar{m}$ *arcs.*

# 4 Extensions

In this section, we study two extensions of the temporal shortest path interdiction problem. The first extension is motivated by [13] and allows for continuous-time availability of arcs. It is shown that even a slight generalization makes it hard to decide whether the nodes $s$ and $t$ can be separated by an interdiction strategy in a temporal graph. The second extension, motivated by [14], imposes an additional constraint on the maximum waiting time in a node. It is shown that the additional constraint does not change the results from Section 3.

## 4.1 Continuous Time Availability of Arcs

In this section, a slightly more general model of temporal graphs is investigated, where the start time $\tau(r)$ of a temporal arc $r$ is not given by one fixed point in time, but rather by a closed interval $[\tau^{\mathrm{l}}(r), \tau^{\mathrm{u}}(r)] =: \tau(r)$. The arc can then be entered at any time $\tau \in \tau(r)$, leading to an arrival time of $\tau + \lambda(r)$ at $\omega(r)$. In the following, we therefore no longer speak of a start time, but of an *availability interval*. The resulting temporal graphs where arcs are available during availability intervals are called *continuous-time temporal graphs*. Note that the class of continuous-time temporal graphs comprises the class of discrete-time temporal graphs, which correspond to continuous-time temporal graphs in which each availability interval only consists of a single point.

While a temporal path in a discrete-time temporal graph is given by a sequence of temporal arcs, the definition has to be slightly adapted in the continuous-time case. A *(continuous-time) temporal path* in a continuous-time temporal graph is a sequence $P = ((r_1, \tau_1), \dots, (r_k, \tau_k))$ of pairs of an arc and a start time with $\tau_i \in \tau(r_i)$ for each $i \in \{1, \dots, k\}$ such that $\omega(r_i) = \alpha(r_{i+1})$ and $\tau_i + \lambda(r_i) \leq \tau_{i+1}$ for each $i \in \{1, \dots, k-1\}$.

We show that, given a continuous-time temporal graph $G$ and a positive integer $B$, it is strongly $\mathcal{NP}$-hard to decide whether a pair $(s, t)$ of nodes can be separated by removing at most $B$ arcs from $G$. This immediately implies that all variants of temporal shortest

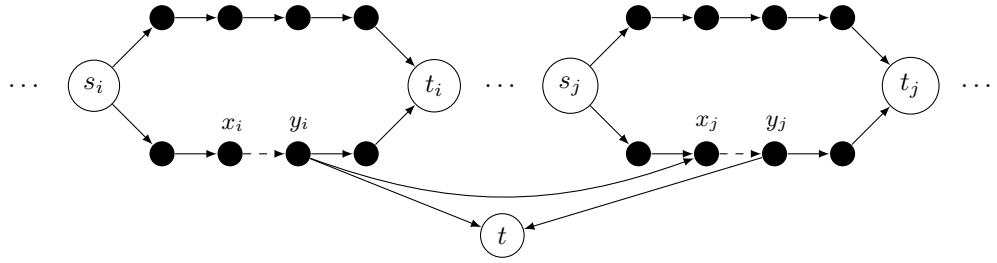



**Figure 2** The gadgets in $G$ for two nodes $i$ and $j$ that are adjacent in $G'$ (where $i < j$). Only the dashed arcs can be interdicted.

path interdiction problems studied in this paper are strongly $\mathcal{NP}$-hard on continuous-time temporal graphs, and that they do not not even admit polynomial-time approximation algorithms with a bounded approximation ratio (unless $\mathcal{P} = \mathcal{NP}$).

The reduction, which is similar to the one presented in [3], is from the well-known (strongly) $\mathcal{NP}$-hard node cover problem, which is defined as follows:

| | |
|---|---|
| INSTANCE: | An undirected (static) graph $G' = (V', E)$ and a positive integer $B' \leq \lvert V' \rvert$ |
| QUESTION: | Is there a subset $\bar{V} \subseteq V'$ of nodes with $\lvert \bar{V} \rvert \leq B'$ such that each edge in $E$ is incident to at least one node in $\bar{V}$? |

Given an instance of node cover, a continuous-time temporal graph $G$ and a budget $B$ are constructed as follows. The budget is chosen as $B := B'$. For the construction of the continuous-time temporal graph $G$, it is assumed without loss of generality that $V' = \{1, \dots, n'\}$. For each node $i \in \{1, \dots, n'\}$, a gadget consisting of two parallel paths of length five is constructed. These paths are referred to as the *upper* and *lower* path of the gadget. The start node of the two paths is referred to as $s_i$ and the end node of the two paths as $t_i$. Further, the start and end node of the third arc in the lower path are referred to as $x_i$ and $y_i$, respectively. All arcs in a gadget, except for the third arc in the lower path, have removal cost $B+1$ and availability interval $[0, 5n']$. The third arc in the lower path has removal cost 1 and availability interval $[5i-4, 5i-3]$, and we identify removing this arc with including node $i$ in the node cover. All arcs in the gadget have traversal time 1.

The gadgets are connected by identifying $t_i$ with $s_{i+1}$ for all $i \in \{1, \dots, n'-1\}$. Further, for an edge $e \in E$ that is incident to the nodes $i$ and $j$ with $i < j$, there exists an arc from $y_i$ to $x_j$ with availability interval $[5i-2, 5i-2]$, traversal time $5(j-i)-2$, and removal cost $B+1$. This arc is called the *shortcut* from $i$ to $j$.

The node $s$ for the instance of temporal shortest path interdiction is $s_1$. The node $t$ is added and, for each $i \in \{1, \dots, n'\}$, there exists an arc from $y_i$ to $t$ with availability interval $[5i-3, 5i-3]$, traversal time 0, and removal cost $B+1$. An illustration of the construction is provided in Figure 2.

To achieve unit removal costs, we can simply replace each arc $r$ by $c(r)$ many identical copies with unit removal cost (as for the proof of Theorem 12 and Corollary 13 for unit removal costs). Note that this conserves the polynomial size of the constructed instance. Using this construction, we prove the following theorem.

▶ **Theorem 18.** *Deciding whether a pair $(s,t)$ of nodes can be separated by removing at most $B$ arcs from a continuous-time temporal graph is strongly $\mathcal{NP}$-complete.*

**Proof.** The problem is clearly in $\mathcal{NP}$ since it can be easily checked in polynomial time whether a given set of at most $B$ arcs separates $s$ from $t$. To show $\mathcal{NP}$-completeness, let $S$ be an interdiction strategy for the constructed instance. Observe that each $s$-$t$-path in $G_S$

traverses at least one shortcut from $i$ to $j$ for some $i, j \in \{1, \dots, n'\}$. This is possible if and only if none of the removable arcs in gadgets $i$ and $j$ are removed by the interdiction strategy. By identifying the interdiction of an interdictable arc in a gadget $i$ with the inclusion of node $i$ in the node cover, it follows that there exists a node cover of size $B'$ $(= B)$ if and only if there exists an interdiction strategy in $G$ that removes at most $B$ arcs. ◀

## 4.2 Waiting Time Constraints

The definition of an $s$-$t$-path provided in Section 2 allows to wait at nodes for any length of time. However, arbitrarily long waiting times are often undesired in real-world problems such as, e.g., packet routing in communication networks. To this end, the problem of finding a $\Delta$-*restless temporal $s$-$t$-path* that cannot wait longer than a given amount of time $\Delta$ in any node except $s$ and $t$ has been defined (see [14]). As shown in [14], deciding whether a *simple* $\Delta$-restless $s$-$t$-path (i.e., a $\Delta$-restless $s$-$t$-path that does not visit any node more than once) exists is strongly $\mathcal{NP}$-hard for any $\Delta \geq 0$.[3] However, in the setting considered here where paths are not required to be simple, this problem is polynomial-time solvable. A Dijkstra-like polynomial-time algorithm for computing *not necessarily simple* restless paths in temporal graphs is presented in [5].

In this section, we show how the time-expanded graph introduced in Section 3.1 can be modified to account for additional waiting time constraints. Further, we show that the complexity of the four versions of temporal shortest path interdiction does not change under additional waiting time constraints.

Within this section, we assume that $s$ has no incoming arcs and $t$ has no outgoing arcs. This assumption does not impose a loss of generality since a shortest $s$-$t$-path (with respect to any of the definitions of "shortest") that uses such an arc could be transformed into one that does not.

For the construction of the time-expanded graph $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$ under waiting time constraints, recall the set $T = \{\phi_1, \dots, \phi_j\}$ of crucial times, which are indexed in increasing order. Similar to the construction of the time-expanded graph in Section 3.1, we introduce a node $(v, \phi_i)$ for every $v \in V$ and $i \in \{1, \dots, j\}$. For $v \in \{s, t\}$ and $i \in \{1, \dots, j-1\}$, there exists an arc from $(v, \phi_i)$ to $(v, \phi_{i+1})$, which represents waiting at $s$ before the start of the path or waiting at $t$ after having arrived. For each arc $r \in R$, an additional node $u_r$ is introduced. Further, there exists an arc in $R^{\text{te}}$ from $(\alpha(r), \tau(r))$ to $u_r$ and an arc from $u_r$ to any node $(\omega(r), \phi)$ with $\phi \in [\tau(r) + \lambda(r), \tau(r) + \lambda(r) + \Delta]$. Traversing the arc from $(\alpha(r), \tau(r))$ to $u_r$ and then the arc from $u_r$ to some node $(\omega(r), \phi)$ represents traversing $r$ in the temporal graph and entering the next arc in the path (if $\omega(r) \neq t$) exactly at time $\phi$. This completes the construction of $H^{\text{te}}$.

To show a one to one correspondence between $s$-$t$-paths in $G$ and $(s, \phi_1)$-$(t, \phi_j)$-paths in $H^{\text{te}}$, note that there are no parallel arcs in $H^{\text{te}}$, which implies that any path in $H^{\text{te}}$ is uniquely given by its trace.

▶ **Observation 19.** *There exists a $\Delta$-restless $s$-$t$-path in $G$ if and only if there exists a $(s, \phi_1)$-$(t, \phi_j)$-path in $H^{\text{te}}$.*

[3] It is worth noting that the definition of temporal graphs in [14] is slightly different. They state that the problem is strongly $\mathcal{NP}$-hard for any $\Delta \geq 1$, but, indeed, the same proof of hardness with a slight modification holds true for the definition of temporal graphs used here for any $\Delta \geq 0$.

**Proof.** Let $P = (r_1, \dots, r_k)$ be a $\Delta$-restless $s$-$t$-path in $G$. Then we claim that the unique path with trace

$$((s, \phi_1), \dots, (s, \tau(r_1)), u_{r_1}, (\omega(r_1), \tau(r_2)), u_{r_2}, (\omega(r_2), \tau(r_3)), \dots, (t, \tau(r_k) + \lambda(r_k)), \dots, (t, \phi_j))$$

is a $(s, \phi_1)$-$(t, \phi_j)$ path in $H^{\text{te}}$. Since, for $v \in \{s, t\}$ and $i \in \{1, \dots, j-1\}$, there exists an arc from $(v, \phi_i)$ to $(v, \phi_{i+1})$, the arcs from $(s, \phi_1)$ to $(s, \tau(r_1))$ and the arcs from $(t, \tau(r_k) + \lambda(r_k))$ to $(t, \phi_j)$ are in $H^{\text{te}}$. Moreover, the arc from $(\alpha(r_i), \tau(r_i))$ to $u_{r_i}$ is in $H^{\text{te}}$ for $i \in \{1, \dots, k\}$, and the arc from $u_{r_i}$ to $(\omega(r_i), \tau(r_{i+1}))$ is in $H^{\text{te}}$ for $i \in \{1, \dots, k-1\}$ since $P$ is $\Delta$-restless.

Conversely let $P'$ be an $(s, \phi_1)$-$(t, \phi_j)$-path in $H^{\text{te}}$. From the construction, it immediately follows that the trace of $P'$ must be of the above form and, by the same arguments as above, it follows that $(r_1, \dots, r_k)$ is a $\Delta$-restless $s$-$t$-path in $G$. ◀

To show that T-LSP-IP and T-EAP-IP remain polynomial-time solvable, we assign removal costs to $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$. For each arc $r \in R$, the (unique) incoming arc of $u_r$ has removal cost $c(r)$. All other arcs have removal cost $B + 1$. With this construction, we observe the following.

▶ **Observation 20.** *There exists an interdiction strategy $S$ such that there does not exist a $\Delta$-restless path in $G_S$ if and only if there exists an interdiction strategy $S'$ separating $s^{\text{te}}$ from $t^{\text{te}}$ in $H^{\text{te}}$.*

Using the algorithm proposed in Sections 3.1 and 3.2 together with the time-expanded graph $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$ under waiting time constraints constructed in this section, it follows that the problems remain polynomial-time solvable under waiting time constraints.

▶ **Theorem 21.** *There exists a polynomial-time algorithm for solving T-LSP-IP and T-EAP-IP under waiting time constraints for each $\Delta \geq 0$.*

We proceed with assessing the complexity of T-SDP-IP and T-STP-IP under waiting time constraints. When taking a closer look at the reduction provided in Section 3.3, every $s$-$t$-path in the constructed instance is 0-restless. This immediately implies that T-SDP-IP under waiting time constraints is strongly $\mathcal{NP}$-hard for every $\Delta \geq 0$. Further, on instances where waiting in any node except $s$ and $t$ is impossible, the problems T-SDP-IP and T-STP-IP are equivalent. This yields the following theorem.

▶ **Theorem 22.** *The problems T-SDP-IP and T-STP-IP are strongly $\mathcal{NP}$-hard under waiting time constraints for each $\Delta \geq 0$.*

To close this chapter, we argue that T-SDP-IP and T-STP-IP are still polynomial-time solvable on extension-parallel temporal graphs under waiting time constraints. To this end, when removing each arc $r$ whose corresponding $s$-$t$-path $P_r$ is not a temporal path from the graph and from the set $\bar{R}$ as in Lemma 15, we additionally check whether $P_r$ is $\Delta$-restless and remove $r$ if this is not the case. Afterwards, the graph contains exactly the $\Delta$-restless temporal $s$-$t$-paths and the algorithm presented in Section 3.3.1 can be used to solve T-SDP-IP and T-STP-IP on extension-parallel temporal graphs under waiting time constraints, which yields the following theorem.

▶ **Theorem 23.** *There exists a polynomial-time algorithm for T-SDP-IP and T-STP-IP on extension-parallel (temporal) graphs under waiting time constraints for each $\Delta \geq 0$.*

It is worth noting that all results presented in this section can easily be adapted to the general case where waiting times are constrained node-wise instead of globally. The only difference to the global case is in the construction of the time-expanded graph, where the node-wise constriction is enforced by the outgoing arcs of the nodes $u_r$ for $r \in R$.

## 5 Conclusion

In this paper, the complexity of four different versions of temporal shortest path interdiction is analyzed. While the latest start and the earliest arrival path interdiction problems are shown to be solvable in polynomial time, the shortest duration and the shortest traversal path interdiction problems are strongly $\mathcal{NP}$-hard. It is particularly interesting that, even though *temporal* shortest path interdiction seems more complex than its static counterpart, which is known to be strongly $\mathcal{NP}$-hard, there are versions of temporal shortest path interdiction problems that are polynomially solvable. We further provide polynomial-time algorithms for the two hard problems on extension-parallel temporal graphs, which can also be transferred to the static shortest path interdiction problem.

An interesting direction for future work could be to study temporal shortest path interdiction problems for other types of modifications than arc removal. For example, one could consider the problem of worsening (or improving) the latest start time, the earliest arrival time, the shortest duration, or the shortest traversal time as much as possible by changing a given number of start times of arcs in a temporal graph.

## References


1 E. C. Akrida, G. B. Mertzios, P. G. Spirakis, and V. Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020. doi:10.1016/j.jcss.2019.08.002.

2 M. O. Ball, B. L. Golden, and R. V. Vohra. Finding the most vital arcs in a network. *Operations Research Letters*, 8(2):73–76, 1989. doi:10.1016/0167-6377(89)90003-5.

3 A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, University of Maryland, 1995.

4 B. M. Behring, A. Rizzo, and M. Porfiri. How adherence to public health measures shapes epidemic spreading: A temporal network model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(4):043115, 2021. doi:10.1063/5.0041993.

5 M. Bentert, A.-S. Himmel, A. Nichterlein, and R. Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):73, 2020. doi:10.1007/s41109-020-00311-0.

6 C. Bentz. On the hardness of finding near-optimal multicuts in directed acyclic graphs. *Theoretical Computer Science*, 412(39):5325–5332, 2011. doi:10.1016/j.tcs.2011.06.003.

7 K. A. Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks*, 28(3):125–134, 1996. doi:10.1002/(SICI)1097-0037(199610)28:3<125::AID-NET1>3.0.CO;2-P.

8 J. Boeckmann and C. Thielen. A $(B+1)$-approximation for network flow interdiction with unit costs. *Discrete Applied Mathematics (online first)*, pages 1–14, 2021. doi:10.1016/j.dam.2021.07.008.

9 F. Brunelli, P. Crescenzi, and L. Viennot. On computing Pareto optimal paths in weighted time-dependent networks. *Information Processing Letters*, 168:106086, 2021. doi:10.1016/j.ipl.2020.106086.

10 B. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.

11 C. Burch, R. Carr, S. Krumke, M. Marathe, C. Phillips, and E. Sundberg. A decomposition-based pseudoapproximation algorithm for network flow inhibition. In D. L. Woodruff, editor, *Network Interdiction and Stochastic Integer Programming*, chapter 1, pages 51–68. Kluwer Academic Press, 2003. doi:10.1007/0-306-48109-X_3.

12 S. Busam, L. E. Schäfer, and S. Ruzika. The two player shortest path network interdiction problem, 2020. arXiv:2004.08338.

13 A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.

14 A. Casteigts, A.-S. Himmel, H. Molter, and P. Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. doi:10.1007/s00453-021-00831-w.

15 S. R. Chestnut and R. Zenklusen. Hardness and approximation for network flow interdiction. *Networks*, 69(4):378–387, 2017. doi:10.1002/net.21739.

16 A. Deligkas and I. Potapov. Optimizing reachability sets in temporal graphs by delaying. *Information and Computation*, 285:104890, 2022. doi:10.1016/j.ic.2022.104890.

17 J. Enright, K. Meeks, G. B. Mertzios, and V. Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021. doi:10.1016/j.jcss.2021.01.007.

18 J. Enright, K. Meeks, and F. Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021. doi:10.1016/j.jcss.2020.08.001.

19 A. Epstein, M. Feldman, and Y. Mansour. Strong equilibrium in cost sharing connection games. In *Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*, pages 84–92, 2007. doi:10.1145/1250910.1250924.

20 T. Fluschnik, H. Molter, R. Niedermeier, M. Renken, and P. Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020. doi:10.1016/j.tcs.2019.03.031.

21 T. Holzmann and J.C. Smith. The shortest path interdiction problem with randomized interdiction strategies: Complexity and algorithms. *Operations Research*, 69(1):82–99, 2021. doi:10.1287/opre.2020.2023.

22 A. Ibiapina, R. Lopes, A. Marino, and A. Silva. Menger's Theorem for temporal paths (not walks), 2022. arXiv:2206.15251.

23 D. Kempke, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the 32nd ACM Symposium on the Theory of Computing (STOC)*, pages 504–513, 2000.

24 L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, G. Rudolf, and J. Zhao. On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems*, 43(2):204–233, 2008. doi:10.1007/s00224-007-9025-6.

25 N. Maack, H. Molter, R. Niedermeier, and M. Renken. On finding separators in temporal split and permutation graphs. *Journal of Computer and System Sciences*, 135:1–14, 2023. doi:10.1016/j.jcss.2023.01.004.

26 O. Michail and P.G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.

27 H. Molter. *Classic Graph Problems Made Temporal – A Parameterized Complexity Analysis*. PhD thesis, Technische Universität Berlin, 2020.

28 H. Molter. The complexity of finding temporal separators under waiting time constraints. *Information Processing Letters*, 175:106229, 2022. doi:10.1016/j.ipl.2021.106229.

29 H. Molter, M. Renken, and P. Zschoche. Temporal reachability minimization: Delaying vs. deleting, 2021. arXiv:2102.10814.

30 P. Mutzel and L. Oettershagen. On the enumeration of bicriteria temporal paths. In *Proceedings of the 15th Annual Conference on Theory and Applications of Models of Computation (TAMC)*, volume 11436 of *Lecture Notes in Computer Science*, pages 518–535, 2019. doi:10.1007/978-3-030-14812-6_32.

31 L. Oettershagen. *Temporal Graph Algorithms*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2022.

**32** J. B. Orlin. Minimum convex cost dynamic network flows. *Mathematics of Operations Research*, 9(2):190–207, 1984. doi:10.1287/moor.9.2.190.

**33** C. Phillips. The network inhibition problem. In *Proceedings of the 25th ACM Symposium on the Theory of Computing (STOC)*, pages 776–785, 1993.

**34** J. A. Sefair and J. C. Smith. Dynamic shortest-path interdiction. *Networks*, 68(4):315–330, 2016. doi:10.1002/net.21712.

**35** J.C. Smith and Y. Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020. doi:10.1016/j.ejor.2019.06.024.

**36** J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi:10.1145/800135.804393.

**37** R. D. Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964. doi:10.1287/opre.12.6.934.

**38** H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016. doi:10.1109/TKDE.2016.2594065.

**39** P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020. doi:10.1016/j.jcss.2019.07.006.

# A Connectivity-Sensitive Approach to Consensus Dynamics

**Bernard Chazelle**
Department of Computer Science, Princeton University, NJ, USA

**Kritkorn Karntikoon**
Department of Computer Science, Princeton University, NJ, USA

## Abstract

The paper resolves a long-standing open question in network dynamics. Averaging-based consensus has long been known to exhibit an exponential gap in relaxation time between the connected and disconnected cases, but a satisfactory explanation has remained elusive. We provide one by deriving nearly tight bounds on the *s-energy* of disconnected systems. This in turn allows us to relate the convergence rate of consensus dynamics to the number of connected components. We apply our results to opinion formation in social networks and provide a theoretical validation of the concept of an *Overton window* as an attracting manifold of "viable" opinions.

**2012 ACM Subject Classification** Mathematics of computing → Combinatoric problems; Theory of computation → Dynamic graph algorithms

**Keywords and phrases** $s$-energy, dynamic networks, relaxation time, multiagent systems

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.10

**Funding** This work was supported in part by NSF grant CCF-2006125.

## 1 Introduction

Consensus dynamics based on local averaging has been the object of considerable attention [5, 8, 9, 10, 11, 21, 26, 31, 32, 33, 34, 36, 37, 39, 40, 41, 42, 45, 48, 47, 50, 54]. This owes largely to the ubiquity of these systems, from flocking and swarming to synchronization, social epistemology, and opinion dynamics [7, 12, 18, 21, 22, 26, 32, 34, 37, 55]. Agents interact across a network by averaging their state variables with those of their neighbors. Under mild conditions, such systems are known to converge to a fixed-point attractor.

When the network is fixed, the dynamics is dual to a Markov chain, which puts a wealth of analytical tools at our disposal. It is well known that convergence within $\varepsilon$ is reached in time $C\log(1/\varepsilon)$, for some parameter $C$ depending only on the graph's size and topology. This bound still holds for time-varying graphs as long as they remain connected at all times. When connectivity is not guaranteed, however, the convergence time shoots up to $C\log(1/\varepsilon)^{n-1}$. This exponential jump has been a puzzling mystery in the field of time-varying network dynamics [13, 26, 34, 50, 51]. Recent works on oblivious message adversaries also exhibit exponential gaps in the time complexity of certain broadcast and consensus problems [17, 23, 24, 25, 57]. The gap is also behind the emergence of hyper-torpid mixing in *Markov influence systems* and the *slow-clock* phenomenon [14].

This paper explains the exponential jump in consensus dynamics by relating it to the number of connected components in the system. The convergence rate is shown to be of the form $C(\log 1/\varepsilon)^m$, where $m < n$ is the maximum number of connected components at any time. We derive quasi-optimal bounds on the parameter $C$. In addition, we look at three important special cases – *reversible*, *expanding*, *random* – and we discuss applications to opinion formation in social networks. The results in this work rely on new *s-energy* bounds of independent interest. The $s$-energy is a generating function designed specifically


© Bernard Chazelle and Kritkorn Karntikoon;
licensed under Creative Commons License CC-BY 4.0

for the analysis of networked averaging systems [12, 13]. Its main purpose is to overcome the technical difficulties one encounters when systems become disconnected and one does not have apriori bounds on how long they might stay so. The difficulty is fundamental: With changing topologies, networked systems cease to have coherent eigenmodes and spectral techniques break down. In other words, when linear algebra fails, the $s$-energy gives us a way out.

### Averaging dynamics

Let $(G_t)_{t=1}^{\infty}$ be an infinite sequence of graphs over the vertex set $[n]$. Each graph has a self-loop. Let $P_t$ be the stochastic matrix of a random walk over $G_t$. By construction, a matrix entry is positive if and only if it corresponds to an edge of $G_t$. Each row sums up to 1 and the diagonal is positive. We assume that the nonzero entries in $P_t$ are at least some fixed $\rho \in (0, 1/2]$.[1] Let $P_{\leq t}$ denote the product $P_t \cdots P_1$. The set of orbits $(P_{\leq t}x)_{t>0}$, over all $x \in \mathbb{R}^n$, forms an *averaging system*, often called *consensus dynamics* in the literature [26]. When all the matrices $P_t = P$ are identical, the map $x \mapsto Px$ is the dual map of the Markov chain $(y \mapsto yP)$ and its convergence time is the chain's *mixing time*. The case of a fixed matrix has been studied exhaustively, so the novelty of the paper comes from the dynamic nature of the networks.

- A *general* averaging system (*genS*) assumes only that the graphs $G_t$ are undirected.
- A *reversible* averaging system (*revS*) is a *genS* whose individual Markov chains $P_t$ are reversible and share the same stationary distribution. This means that $P_t = \mathrm{diag}(q)^{-1}M_t$, where $M_t$ is symmetric with nonzero entries at least 1 and $q = M_t\mathbf{1} \preceq \mathbf{1}/\rho$.
- An *expanding* averaging system (*expS*) is a *revS* where $q = d\mathbf{1}$ and the connected components of each $G_t$ are $d$-regular *expanders*. Recall that a $d$-regular expander is a graph of degree $d$ such that, for any set $X$ of at most half of the vertices, we have $|\partial X| \geq h|X|$, where $\partial X$ is the set of edges with exactly one vertex in $X$; the factor $h$ is called the Cheeger constant.
- A *random* averaging system (*ranS*) assumes that the graphs $G_t$ are $d$-regular and random.

### Our results

Given $x \in [0, 1]^n$, each point of the orbit $(P_{\leq t}\, x)_{t>0}$ corresponds to an embedding of the graph $G_t$ over the reals. Let $T_\varepsilon$ be the number of timesteps $t$ at which $G_t$ has an embedded edge of length at least $\varepsilon > 0$. We denote by $T_{m,\varepsilon}$ the maximum value of $T_\varepsilon$ over all graph sequences $(G_t)_{t>0}$ such that no graph has more than $m$ connected components.[2] We use superscripts to distinguish among the general, reversible, expanding, and random cases: $T^{\mathrm{GEN}}, T^{\mathrm{REV}}, T^{\mathrm{EXP}}$ and $T^{\mathrm{RAN}}$ respectively.

For simplicity, $T^{\mathrm{REV}}, T^{\mathrm{EXP}}$ and $T^{\mathrm{RAN}}$ do not assume that the initial diameter is bounded by 1 but, rather, that the initial variance is. Here we define the (scaled) variance as $\|x - \hat{x}\,\|_q^2$, where $x$ is shorthand for $x(1)$ and (i) $\hat{x}$ is the mean initial position $\|q\|_1^{-1}\langle x, \mathbf{1}\rangle_q\mathbf{1}$; (ii) $\|x\|_q^2 := \langle x, x\rangle_q$; and (iii) $\langle x, y\rangle_q := \sum_i q_i x_i y_i$.

---

[1] If $\rho > 1/2$, the only edges of $G_t$ have to be self-loops, which is of no interest.

[2] Since the systems always converge to a fixed-point attractor, the reader might wonder why we do not define $T_\varepsilon$ as the time past which no edge length exceeds $\varepsilon$. This would not work because an adversary could always insert the identity matrix repeatedly to delay convergence at will and push $T_\varepsilon$ to infinity.

Note that unit diameter implies a variance of at most $n/\rho$ and, conversely, unit variance implies a diameter bounded by 2. Thus, the following bounds can be easily scaled to accommodate either assumption about initial conditions.

▶ **Theorem 1.** *For some constant $c > 0$ and any positive $\varepsilon$ small enough,*

$$\begin{cases} T^{\text{GEN}}_{m,\varepsilon} \le c(1/\rho)^{n-1}\left(mn\log\frac{1}{\varepsilon}\right)^m \\ T^{\text{REV}}_{m,\varepsilon} \le c\left(\frac{n^2}{\rho}\log\frac{1}{\varepsilon}\right)^m \quad [15] \\ T^{\text{EXP}}_{m,\varepsilon} \le c\left(\frac{d^3mn}{h^2}\log\frac{1}{\varepsilon}\right)^m \quad \& \quad T^{\text{EXP}}_{1,\varepsilon} \le \frac{cd^3}{h^2}\log\frac{1}{\varepsilon} \\ \mathbb{E}\,T^{\text{RAN}}_{1,\varepsilon} \le c\log\frac{1}{\varepsilon}. \end{cases}$$

**Proof.** The upper bound on $T^{\text{REV}}_{m,\varepsilon}$ was proven in [15] and is mentioned here for completeness. Note that the case $m = 1$ of *genS* and *revS* recover the classic mixing times for Markov chains ($P_t = P$), in particular the polynomial vs. exponential gap between general and reversible chains. We also rediscover the logarithmic bound for expanders ($m = 1$). Previous work addressed only the cases $m = 1$ or $m = n - 1$. What made any connectivity-sensitive extension challenging is that the proof techniques for the $s$-energy do not seem to generalize. It is often possible to set up recurrence relations but these are too coarse to deliver good upper bounds. Intricate multiscale amortization arguments were used to overcome these limitations. In a surprising turn, we show how to rescue the divide-and-conquer approach via a new linearization technique. Before we turn to the $s$-energy $\mathcal{E}_{m,s}$, we need to mention its relevance: Theorem 1, indeed, follows from combining the corresponding $s$-energy bounds with the inequality

$$T_{m,\varepsilon} \le \inf_{0<s\le 1} \varepsilon^{-s}\mathcal{E}_{m,s}. \tag{1}$$

The idea is to provide upper bounds on $\mathcal{E}_{m,s}$ for each of the four cases: general, reversible, expanding, and random. Proving Theorem 1 is then a matter of choosing $s$ to minimize the right-hand side in (1). This step is straightforward calculus and, hence, omitted. ◀

The bounds in Theorem 1 are very general and can be applied to countless instances of real-world dynamics (swarming, flocking, polarization, power grid sync, firefly flashing, etc. [12]). We conclude this work in Section 3 with an application of our ideas to opinion formation in social networks. We extend the model to include directed edges so as to capture both evolving and fixed sources of information. We show that, while all opinions might keep changing forever, they will inevitably land in the convex hull of the fixed sources. Furthermore, we bound the time at which this must happen. Our result is a quantitative validation of the *Overton window* as an attracting manifold of "viable" opinions [4, 6, 19, 27, 46].

## 2 New Bounds on the $s$-Energy

Let $(G_t)_{t=1}^{\infty}$ be an infinite sequence of graphs over the vertex set $[n]$. A vertex is also called an *agent*. Each graph is embedded in $\mathbb{R}$ and we denote by $x_i(t)$ the position of agent $i$. The union of the embedded edges of $G_t$ forms disjoint intervals, called *blocks*. Let $l_1, \ldots, l_k$ be the lengths of these blocks and put $E_{s,t} = \sum_{i=1}^{k} l_i^s$, with $s \in (0, 1]$.[3] We define the *$s$-energy* $E_s = \sum_{t\ge 1} E_{s,t}$ and we denote by $\mathcal{E}_{m,s}$ the supremum of $E_s$ , over all initial agent positions $x \in [0, 1]^n$, under the constraint that $G_t$ should have at most $m$ connected components.

[3] For example, if $G_t$ consists of three edges embedded as $[0.1, 0.3], [0.2, 0.4]$ and $[0.7, 0.8]$, then there are two blocks $[0.1, 0.4], [0.7, 0.8]$ and $E_{s,t} = (0.3)^s + (0.1)^s$.

## 2.1 General averaging systems

We begin by stating our bounds on the $s$-energy of any *genS* with at most $m$ connected components. As stated above, we assume that the initial diameter $\Delta$ of the vertex positions is 1. If it is not, it suffices to multiply the bounds by a factor of $\Delta^s$.

▶ **Theorem 2.** *$\mathcal{E}^{\text{GEN}}_{m,s} \leq (c/s)^m (1/\rho)^{n-1}$, for any $s \in (0,1]$, where $c = O(mn)$. For $s = 1$, the bound can be improved to $\mathcal{E}^{\text{GEN}}_{m,1} \leq 3en(1/\rho)^{\lfloor n/2 \rfloor}$.*

### Twist systems

A *genS* is a special case of a *twist system* [13]. The latter is easier to analyze so we turn our attention to it. Relabel the agents so their positions $x_1 \leq \cdots \leq x_n$ appear in sorted order at time $t$. A twist system moves them to positions $y_1 \leq \cdots \leq y_n$ at time $t+1$ in such a way that

$$(1-\rho)x_u + \rho x_{\min\{i+1,v\}} \leq y_i \leq \rho x_{\max\{i-1,u\}} + (1-\rho)x_v, \tag{2}$$

for any $i$ in $[u,v]$ and $y_i = x_i$ otherwise. We repeat this step indefinitely. Twist systems are highly nondeterministic. At each step, a new interval $[u,v] \subseteq [n]$, called a *block*, is picked and the agents' motion is only constrained by (2) and the need to maintain their ranks (ie, agents never cross).

For the purposes of this work, we extend the concept to *$m$-twist systems* by stipulating, at each time $t$, a partition of $[n]$ into up to $m_t$ blocks $[u_{t,l}, v_{t,l}]$ $(1 \leq l \leq m_t \leq m)$. Each agent is now subject to (2) within its own enclosing block. We define the $s$-energy $E^{\text{TW}}_s$ of a twist system as we did with a *genS* by adding together the $s$-th powers of all the block lengths. We use the same notation with the addition of the superscript TW.

▶ **Lemma 3.** *A genS with at most $m$ connected components at any time can be interpreted as an $m$-twist system with the same $s$-energy.*

**Proof.** Fix a *genS* and let $(x_i)_{i=1}^n$ and $(y_i)_{i=1}^n$ be the positions of the agents at times $t$ and $t+1$, given in nondecreasing order. We denote by $x'_i$ the position of agent $i$ at time $t+1$. Let $[x_u, x_v]$ be a block of the *genS* at time $t$. Pick $k < v$ and write $z = \rho x_k + (1-\rho)x_v$. All the diagonal elements of $P_t$ are at least $\rho$; hence $x'_i \leq z$, for all $i \leq k$, and $y_k \leq z$. In fact, the inequality even holds for $i = k+1$: Indeed, the embedded edges of $G_t$ cover all of $[x_u, x_v]$, so at least one of them, call it $(l,r)$, must join $[u,k]$ to $[k+1,v]$; hence $x'_r \leq z$. Our claim follows. This proves that, for all $i \in (u,v]$, $y_i \leq \rho x_{\max\{i-1,u\}} + (1-\rho)x_v$. We omit the case $i = u$ and the mirror-image inequality, which repeat the same argument. Summing up all the powers $(x_v - x_u)^s$ shows the equivalence between the two $s$-energies. ◀

**Proof of Theorem 2.** We may assume that the agents stay within $[0,1]$. We begin with showing the bound $\mathcal{E}^{\text{GEN}}_{m,1} \leq \mathcal{E}^{\text{TW}}_{m,1} \leq 3en(1/\rho)^{\lfloor n/2 \rfloor}$.

Case $s = 1$

We prove a stronger result by bounding $K_t(z) := \sum_{k=1}^n \left(x_{v(k)} - x_k\right) z^k$, where $v(k) = v_{t,l}$ for $l$ such that $k \in [u_{t,l}, v_{t,l}]$. As usual, $0 \leq x_1 \leq \cdots \leq x_n \leq 1$ denotes the sorted positions of the agents at time $t$; we omit $t$ for convenience but it is understood throughout. We define the *weighted* 1*-energy* $K(z) = \sum_{t>0} K_t(z)$ and, finally, $\mathcal{K}(z) = \sup K(z)$. As long as $z \geq 1$, the 1-energy is obviously dominated by its weighted version. We improve this crude bound via a symmetry argument:

▶ **Lemma 4.** *For any $z \geq 1$, $\mathcal{E}^{\text{TW}}_{m,1} \leq 2z^{-\nu}\,\mathcal{K}(z)$, where $\nu = \lceil n/2 \rceil$.*

**Proof.** We define the mirror image of $K_t$ as $\bar{K}_t(z) = \sum_{k=1}^{n} \left(x_k - x_{u(k)}\right) z^{n-k+1}$, where $u(k)$ is the left counterpart of $v(k)$. We have

$$\begin{aligned} E^{\text{TW}}_{1,t} &\leq \sum_{k\leq\nu} \left(x_k - x_{u(k)}\right) + \sum_{k\geq\nu} \left(x_{v(k)} - x_k\right) \\ &\leq z^{-\nu} \sum_{k\leq\nu} \left(x_k - x_{u(k)}\right) z^{n-k+1} + z^{-\nu} \sum_{k\geq\nu} \left(x_{v(k)} - x_k\right) z^k \\ &\leq z^{-\nu}\big(\bar{K}_t(z) + K_t(z)\big). \end{aligned}$$

Because $\mathcal{K}(z) = \sup K(z)$, the lemma then follows by summing up all $t > 0$. ◀

We define the polynomial $P_t(z) = \sum_{k=1}^{n} x_k z^k$ for $z > 1/\rho$ and exploit two simple but surprising facts: $P_t(z)$ cannot increase over time;[4] and, at each step, the drop from $P_t(z)$ to $P_{t+1}(z)$ is at least proportional to $K_t(z)$. Thus, we develop a discrete version of the inference: $dP_t/dt \leq -cK_t$ implies

$$\int_{t\geq 1} cK_t \leq -\int_{t\geq 1} \frac{dP_t}{dt} \leq P_1\,.$$

▶ **Lemma 5.** *For any $z > 1/\rho$, $P_t(z) - P_{t+1}(z) \geq (\rho z - 1)K_t(z)$.*

**Proof.** The inequality is additive in the number of blocks so we can assume there is a single one $[u, v]$ at time $t$. Using the notation of (2), we have $y_k \leq \rho x_{\max\{k-1,u\}} + (1-\rho)x_v$; hence

$$\begin{aligned} P_t(z) - P_{t+1}(z) = \sum_{k=u}^{v} (x_k - y_k) z^k &\geq \sum_{k=u}^{v} \left(x_k - \rho x_{\max\{k-1,u\}} - (1-\rho)x_v\right) z^k \\ &\geq (\rho-1)(x_v - x_u)z^u + \sum_{k=u+1}^{v} \rho(x_v - x_{k-1})z^k - \sum_{k=u+1}^{v} (x_v - x_k)z^k \\ &\geq (\rho-1)(x_v - x_u)z^u + \sum_{k=u}^{v-1} \rho z(x_v - x_k)z^k - \sum_{k=u+1}^{v} (x_v - x_k)z^k \\ &\geq (\rho-1)(x_v - x_u)z^u + \sum_{k=u}^{v} (\rho z - 1)(x_v - x_k)z^k + (x_v - x_u)z^u \\ &\geq (\rho z - 1)K_t(z) + \rho(x_v - x_u)z^u. \end{aligned}$$

◀

The lemma implies that

$$(\rho z - 1)K(z) = \sum_{t>0} (\rho z - 1)K_t(z) \leq P_1(z) \leq \sum_{k=1}^{n} z^k = \frac{z^{n+1} - z}{z - 1}\,. \tag{3}$$

[4] Recall that $x_k$ depends on $t$. Note also that, among the $n$ agents, rightward motion within $[0, 1]$ might greatly outweigh the leftward kind. Thus, if most of the $x_i$'s keep growing, how can $P_t(z)$ not follow suit? The point is that $P_t(z)$ puts weights exponentially growing on the right, so their leftward motion, outweighed as it might be, will always dominate with respect to $P_t(z)$. This balancing act between left and right motion is the core principle of twist systems.

With $z = (1+\varepsilon)/\rho$, $\varepsilon = 1/(n-\nu+1)$, and $n > 2$, we find that

$$z^{-\nu}K(z) \le \frac{z^n - 1}{(z-1)(\rho z - 1)z^{\nu-1}} \le 2\rho^{\nu-n}e^{(n-\nu+1)\varepsilon}/\varepsilon \le 2e(n-\nu+1)\rho^{\nu-n} \le \frac{3en}{2}\rho^{\nu-n}.$$

The case $s = 1$ of Theorem 2 follows immediately from Lemma 4. Finally, for $n = 2$, we verify that $\mathcal{E}^{\text{TW}}_{m,1} = \sum_{k\ge 0}(1-2\rho)^k = 1/2\rho$.

Case $s < 1$

The previous argument relied crucially on the linearity of the 1-energy. If $s < 1$, the $s$-energy gives more relative weight to small lengths, so we need a different strategy to keep the scales separated. We omit the superscript TW below but it is understood. We use a threshold $\delta$ which, though set to $1/3$, is best kept as $\delta$ in the notation.

**A recurrence relation.** Let $T_\delta$ be the number of steps at which the diameter remains above $1-\delta$; note that these steps are consecutive and $T_\delta$ might be infinite. By scaling, we find that $E_s \le F_s + (1-\delta)^s E_s$, where $F_s = \sum_{t\le T_\delta} E_{s,t}$. Since $(1-\delta)^s \le 1 - \delta s$, for $\delta, s \in [0,1]$, we have

$$E_s \le (\delta s)^{-1} F_s. \tag{4}$$

If $m = 1$, then $(1-\delta)F_s \le (1-\delta)T_\delta \le F_1 \le \mathcal{E}_{1,1}$. Thus, by (4) and the previous section,

$$\mathcal{E}_{1,s} \le \frac{3en}{\delta(1-\delta)}(1/s)(1/\rho)^{\lfloor n/2\rfloor}. \tag{5}$$

**The case $m > 1$.** Fix $t \le T_\delta$; if $m_t > 1$, let $j$ maximize $x_{j+1}(t) - x_j(t)$ over all $j = v_i$ and $i < m_t$ (break ties by taking the smallest $j$). This corresponds to the maximum distance between consecutive blocks. For this reason, we call $(j, j+1)$ the *max-gap* at time $t$. We say that $t$ is *ungapped* if $m_t = 1$ or $x_{j+1}(t) - x_j(t) \le \delta/m$; it is *gapped* otherwise. Assuming that $t$ is gapped, let $(j, j+1)$ be its max-gap and write $\zeta_t = \min_k\{\, t < k \le T_\delta \mid \exists\, l : u_{k,l} \le j < v_{k,l} \,\}$: If the set is empty, we set $\zeta_t = T_\delta$; else $l$ is unique and we denote it by $l_t$. We call the interval $[t, \zeta_t]$ a *span* and the block $l_t$, if it exists, its *cap*. We note that $x_{j+1}(k) - x_j(k)$ cannot decrease during the times $k = t, \dots, \zeta_t$. This shows that a cap covers a length greater than $\delta/m$.

We begin with a few words of intuition. The energetic contribution of an ungapped time $t$ is easy to account for: It is at most $m$. On the other hand, the 1-energy is at least the diameter minus the added length of the gaps between blocks, which amounts to at least $1 - 2\delta \ge 1/3$; in other words, $E_{s,t} \le 3mE_{1,t}$. Summing up over all ungapped times and plugging in our bound for $s = 1$ gives us the desired result. Accounting for gapped times is more difficult, as it requires dealing with small scales. If we had only one span, we could simply split the system into two decoupled subsystems and set up a recurrence relation. The problem is that the presence of $k$ capped spans would force us to repeat the recursion $k-1$ times. With no apriori bound on $k$, this approach is not too promising. Instead, we make a bold move: We argue that, because a cap is longer than $\delta/m$, its own 1-energy contribution (ie, its length) is large enough to "pay" for the $s$-energy of its entire span. This is not quite right, of course, but one can fix the argument by using the weighted 1-energy of the cap and upscaling it suitably. Once again, this reduces the problem to the case $s = 1$, so our method is, in effect, a linearization. Here is the proof.

**Proof.** We partition the times between 1 and $T_\delta$ into two subsets $G$ and $U := [1, T_\delta] \setminus G$, each one supplied with its own energetic accounting scheme. We form $G$ by greedily extracting a maximal set of nonoverlapping spans and taking their union.

1. $G \leftarrow \emptyset$ and $t' \leftarrow 1$;
2. **if** $t \leftarrow \min\{\text{ gapped } i \,|\, t' \le i \le T_\delta\}$ exists
3. **then** $G \leftarrow G \cup \{\, i \,|\, t \le i \le \zeta_t \,\}$;
4. **if** $\zeta_t < T_\delta$ **then** $t' = \zeta_t + 1$; go to 2;

We postulate that, for any $0 < s \le 1$, and any number of agents $j \le n$,

$$\mathcal{E}_{m,s} \le c_m (1/s)^m (1/\rho)^{j-1}, \tag{6}$$

and we derive a recurrence relation for $c_m$ (for given $n$).

- *Accounting for $G$:* In line 3, let $k = \zeta_t$ and $(j, j+1)$ be the corresponding max-gap. Suppose that the span $[t, k]$ is capped. The absence of an interval including $j$ and $j+1$ during $[t, k-1]$ implies that $\sum_{t \le l \le k} E_{s,l} \le L + R + m$, where $L$ and $R$ denote the $s$-energy of systems with at most $m-1$ connected components. For reasons we address below, we may assume that $L$ is dominant; hence $R \le L \le \mathcal{E}_{m-1,s}$.[5] It follows that

$$\sum_{l=t}^{k} E_{s,l} \le 2\mathcal{E}_{m-1,s} + m \le 3c_{m-1}(1/s)^{m-1}(1/\rho)^{j-1}. \tag{7}$$

  Note that we (safely) assume $c_{m-1} \ge m$. Using the shorthand $v$ for $v_{k,l_t}$, we have $x_v(k) - x_j(k) \ge x_{j+1}(k) - x_j(k) \ge x_{j+1}(t) - x_j(t) > \delta/m$. We add the artificial multiplier $x_v(k) - x_j(k)$ to (7) to make the right-hand side resemble $K(z)$. Recall that $\delta = 1/3$; assuming that $z > 1/\rho$ from now on, we have

$$\sum_{l=t}^{k} E_{s,l} \le B\big(x_v(k) - x_j(k)\big) z^j, \quad \text{with } B = 9c_{m-1} m\rho (1/s)^{m-1} \tag{8}$$

  The set $G$ is a union of spans. If $\zeta_t = T_\delta$, the last span might not be capped. If so, remove it from $G$ and call the resulting set $G'$. Summing up, we find that $\sum_{t \in G'} E_{s,t} \le B \sum_{t \in G'} K_t(z)$. If the last span is uncapped then $\zeta_t = T_\delta$ and no block contains both $j$ and $j+1$ in the span $[t, T_\delta]$. The $s$-energy expended in that span is thus of the form $L + R \le 2\mathcal{E}_{m-1,s}$.
- *Accounting for $U$:* Only ungapped times belong to $U$, so the 1-energy at time $t \in U$ is at least $1 - \delta - (m_t - 1)\delta/m \ge 1/3$. On the other hand, $E_{s,t} \le m \le 3mE_{1,t} \le 3m\rho K_t(z) \le BK_t(z)$.

Set $z = (1+\varepsilon)/\rho$, for $\varepsilon > 0$. Putting all of our bounds together, we have

$$\begin{aligned} F_s = \sum_{t \le T_\delta} E_{s,t} &\le B \sum_{t \in G'} K_t(z) + 2\mathcal{E}_{m-1,s} + B \sum_{t \in U} K_t(z) \\ &\le 2c_{m-1}\,(1/s)^{m-1}(1/\rho)^{n-1} + BK(z). \end{aligned}$$

[5] The inequality relies on the (easy) fact that the maximum $s$-energy grows monotonically with the number of agents. This is not even needed, however, if we redefine $\mathcal{E}_{m,s}$ as the maximum $s$-energy over all systems with *at most* $n$ agents and then reason with the value $n' \le n$ that achieves the maximum.

Actually, the exponent to $1/\rho$ can be reduced to $n-2$, but this is immaterial. By (3),

$$BK(z) \leq 18c_{m-1}(1/s)^{m-1}(1/\rho)^{n-1}e^{(n+1)\varepsilon}(m/\varepsilon).$$

Setting $\varepsilon = 1/(n+1)$ gives us, for some constant $d > 0$,

$$F_s \leq c_{m-1}(dmn)(1/s)^{m-1}(1/\rho)^{n-1}.$$

We tie up the loose ends by arguing that it was legitimate to assume that $L \geq R$. The point is that individual values of $L$ and $R$ do not matter: only their sums do. Thus, if the $R$s outweigh the $L$s, we restore the dominance of the $L$s by flipping the system around. Finally, by (4), $E_s \leq (3/s)F_s$; and so, by (5), Theorem 2 follows from the recurrence: $c_1 = O(n)$ and $c_m \leq 3dmnc_{m-1}$, for $m > 1$. ◀

◀

### Lower bounds for twist systems

We begin with the case $m = 1$. Assume that $n = 2k+1$. At time $t = 1$, we have $x_k = -x_{-k} = 1/2$ and $x_i = -x_{-i} = \frac{1}{2}(1-\rho^i)$, for $0 \leq i < k$ and $\rho$ small enough. For $t > 1$, we set $x_i(t) = (1-\rho^k)x_i(t-1)$. The agents are labeled $-k, \ldots, k$ from left to right. It is easily verified that this constitutes a twist system for the block $[-k, k]$ with initial unit diameter. The $s$-energy $E$ is $\big(1-\rho^k\big)^s E + 1$ so, for constant $c > 0$,

$$E \geq (c/s)(1/\rho)^{\lfloor n/2 \rfloor}. \tag{9}$$

If $n = 2k$, we set $x_k = -x_{-k} = 1/2$ and $x_i = -x_{-i} = \frac{1}{2}(1-2\rho^i)$, for $1 \leq i < k$. For $t > 1$, we set $x_i(t) = (1-2\rho^k)x_i(t-1)$ and rederive (9).

For the general case, we describe the evolution of an $m$-block twist system with $n$ agents, and denote its $s$-energy by $F(n, m)$: It is assumed that $n-1$ agents are positioned at 0 at time 1 and the last one is at position 1. If $m = 1$, we apply the previous construction after shifting the initial interval from $[-0.5, 0.5]$ to $[0, 1]$. The initial positions still do not match, but we note that, in a single step, we can move the agents anywhere we want in the interval $[\rho, 1-\rho]$ while respecting the constraints of a twist system. This gives us $F(n, 1) = 1 + (1-2\rho)^s E$. By adjusting the constant $c$ in (9), the same lower bound still holds.

For $m > 1$, at time 1, we move the agents $n-1$ and $n$ to positions $\rho$ and $1-\rho$, respectively, and we leave the others (if any) at position 0. We then use an $(m-1)$-block twist system recursively for the agents $1, \ldots, n-1$. This brings these agents to a common position[6] in $[0, \rho]$. This gives us the recurrence relation: $F(n, m) \geq 1 + \rho^s F(n-1, m-1) + (1-2\rho)^s F(n, m)$; hence, by induction, for constant $c > 0$,

$$F(n, m) \geq (c/s)^m \rho^{(m-1)s}(1/\rho)^{\lfloor (n+m-1)/2 \rfloor}. \tag{10}$$

The $s$-energy is often used for small $s$, so we state the case of $s = O\big(1/m \log \frac{1}{\rho}\big)$, which matches the bound of Theorem 2 for $m = n-1$.

▶ **Theorem 6.** *$\mathcal{E}^{\text{TW}}_{m,s} \geq (c/s)^m(1/\rho)^{\lfloor (n+m-1)/2 \rfloor}$, for constant $c > 0$, small enough $\rho$ and $s = O\big(1/m \log \frac{1}{\rho}\big)$.*

[6] To keep the time finite, we can always force completion in a single step once the agents are sufficiently close to each other.

## 2.2 Expanding averaging systems

In a *revS*, the stochastic matrices $P_t$ are of the form $P_t = \mathrm{diag}(q)^{-1}M_t$, where $M_t$ is symmetric with nonzero entries at least 1 and $q = M_t\mathbf{1} \preceq \mathbf{1}/\rho$. We verify that $q$ is the common (dominant) left-eigenvector. We revisit the definition of $\mathcal{E}_{m,s}$ to include only the reversible averaging systems of unit variance $\|x - \hat{x}\|_q^2 = 1$, where $\hat{x} = q\langle x, \mathbf{1}\rangle$. We denote the $s$-energy by $\mathcal{E}_{m,s}^{\mathrm{REV}}$. The following result is already known. (Note that, if the variance is not one, it suffices to multiply the upper bound by $\|x - \hat{x}\|_q^s$.)

▶ **Theorem 7** ([15]). *$\mathcal{E}_{m,s}^{\mathrm{REV}} \le (cn^2/\rho s)^m$, for any $s \in (0,1]$ and constant $c > 0$.*

A $d$-regular expander with Cheeger constant $h$ is a graph of degree $d$ such that, for any set $X$ of at most half the vertices, we have $|\partial X| \ge h|X|$, where $\partial X$ is the set of edges with exactly one vertex in $X$. We say that $G = (V, E)$ is a $d$-regular *$m$-expander* if it has at most $m$ connected components. Recall that an expanding averaging system (*expS*) is a *revS* consisting of $d$-regular $m$-expanders. Each nonzero entry in $M_t$ is equal to 1 and $q = d\mathbf{1}$. We redefine the $s$-energy to include only *expS* of unit variance with at most $m$ connected components and denote it by $\mathcal{E}_{m,s}^{\mathrm{EXP}}$. Adding the expanding assumptions cancels the dependency on $n$ in the case $m = 1$. More generally, we prove the following:

▶ **Theorem 8.** *$\mathcal{E}_{m,s}^{\mathrm{EXP}} \le (c/s)^m$, for any $s \in (0,1]$, where $c = O(d^3/h^2)$ for $m = 1$ and $c = O(d^3mn/h^2)$ for $m > 1$.*

We begin the proof with a lower bound on the Dirichlet form that exploits the expansion of a $d$-regular expander with Cheeger constant $h$. This is known as *Cheeger's inequality*. We include the proof below for completeness.

▶ **Lemma 9.** *If $G = (V, E)$ is connected, then $\sum_{(i,j)\in E}(x_i - x_j)^2 \ge b(h/d)^2\|x - \hat{x}\|_q^2 \ge b(h\Delta)^2/2d$, for constant $b > 0$, where $\Delta$ is the diameter of the agent positions $x_1, \ldots, x_n$.*

**Proof.** All of the ideas in this proof come from [3, 52]. The inequality is invariant under shifting and scaling, so we may assume that $\hat{x} = \mathbf{0}$ and $\|x\|_2 = 1$. Relabel the coordinates of $x$ so they appear in nonincreasing order, and define $y \in \mathbb{R}^n$ such that $y_i = \max\{x_i, 0\}$. Let $\alpha = \mathrm{argmax}_k(y_k > 0)$ and $\beta = \min\{\alpha, \lfloor n/2\rfloor\}$. By switching $x$ into $-x$ if necessary,[7] we can always assume that $\|y\|_2^2 > c := 1/6$ if $\alpha = \beta$, and $\|y\|_2^2 \ge 1 - c$ if $\alpha > \beta$. By Cauchy-Schwarz, $(y_i + y_j)^2 \le 2(y_i^2 + y_j^2)$; hence,

$$\sum_{(i,j)\in E}\left|\, y_i^2 - y_j^2\,\right| = \sum_{(i,j)\in E}(y_i + y_j)|y_i - y_j| \le \sqrt{\sum_{(i,j)\in E}(y_i + y_j)^2 \sum_{(i,j)\in E}(y_i - y_j)^2}$$
$$\le \sqrt{\sum_i 2dy_i^2 \sum_{(i,j)\in E}(y_i - y_j)^2} \le \sqrt{\sum_{(i,j)\in E} 2d(y_i - y_j)^2} \le \sqrt{\sum_{(i,j)\in E} 2d(x_i - x_j)^2}. \tag{11}$$

By the expansion property of $G$, summation by parts yields

$$\sum_{(i,j)\in E}\left|\, y_i^2 - y_j^2\,\right| \ge \sum_{k=1}^{\lfloor n/2\rfloor} hk\big(y_k^2 - y_{k+1}^2\big) + \sum_{k=\lfloor n/2\rfloor+1}^{n-1} h(n-k)\big(y_{k+1}^2 - y_k^2\big) = h\left(\|y\|_2^2 - ny_{\lfloor n/2\rfloor+1}^2\right). \tag{12}$$

[7] Intuitively, by changing all signs if necessary, we force the minority sign among the coordinates of $x$ to be positive unless their contribution to the norm of $x$ is too small.

Suppose that $\alpha > \beta = \lfloor n/2 \rfloor$. It follows from $\sum_{i=1}^{n} x_i = 0$ that $\sum_{i=\alpha+1}^{n} |x_i| = \|y\|_1 \geq (\beta+1) y_{\beta+1}$. By Cauchy-Schwarz, this yields $n y_{\beta+1}^2/4 \leq \sum_{i=\alpha+1}^{n} x_i^2 = 1 - \|y\|_2^2$, and by (12), $\sum_{(i,j)\in E} \left| y_i^2 - y_j^2 \right| \geq (1-5c)h = ch$. If $\alpha = \beta$, then $y_{\lfloor n/2 \rfloor + 1} = 0$ and $\sum_{(i,j)\in E} \left| y_i^2 - y_j^2 \right| \geq h\|y\|_2^2 > ch$. Applying (11) shows that $\sum_{(i,j)\in E} (x_i - x_j)^2 \geq (ch)^2/2d$, which gives us the first inequality of the lemma. The second one follows from the fact that the interval $[a, b]$ enclosing the vertex positions contains 0. By Cauchy-Schwarz, $1 = \|x\|_2^2 \geq a^2 + b^2 \geq \frac{1}{2}(b + |a|)^2 = \Delta^2/2$, and the proof is complete. ◀

Let $G_{\leq t}$ be the graph obtained by adding all the edges from $G_1, \ldots, G_t$. Let $m_t \leq m$ be the number of connected components in $G_t$, and let $\Delta_{t,i}$ denote the diameter of the $i$-th component of $G_t$ (labeled in any order). Let $t_1, \ldots, t_c$ be the times $t > 1$ at which the addition of $G_t$ reduces the number of components in $G_{\leq t-1}$. If no such times exist, set $c = t_c = 1$.

▶ **Lemma 10.** *If $G_{\leq t_c}$ is connected, then $\sum_{t \leq t_c} \sum_{i=1}^{m_t} \Delta_{t,i}^2 \geq \frac{1}{2dmn} \|x - \hat{x}\|_q^2$.*

**Proof.** At any time $t_k > 1$, the drop $d_k$ in the number of components can be achieved by $d_k$ (or fewer) components in $G_{t_k}$. We collect the intervals spanned by these components into a set $F$, to which we add the intervals for the components of $G_1$; thus $|F| < 2m$. A simple convexity argument (omitted) shows that the union of the intervals in $F$ coincides with the interval $[a, b]$ enclosing the $n$ vertices at time 1; so the lengths $l_1, \ldots, l_{|F|}$ of the intervals in $F$ sum up to at least $b - a$. By Cauchy-Schwarz, $\sum_{t \leq t_c} \sum_{i=1}^{m_t} \Delta_{t,i}^2 \geq \sum_{i=1}^{|F|} l_i^2 \geq (b-a)^2/(2m-1)$. The lemma follows from $\|x - \hat{x}\|_q^2 \leq \|q\|_1 (b-a)^2 \leq dn(b-a)^2$.

If we define the variant of the Dirichlet form, $D_t = \sum_i \max_{j:\,(i,j)\in E_t} \big(x_i(t) - x_j(t)\big)^2$, we know from [15] that, for any $x = x(1) \in \mathbb{R}^n$,

$$\|P_t\, x\|_q^2 \leq \|x\|_q^2 - \frac{D_t}{2}\,.$$

It follows that $\|x\|_q^2 - \|x(t_c+1)\|_q^2 \geq \frac{1}{2} \sum_{t \leq t_c} D_t \geq \frac{1}{d} \sum_{t \leq t_c} \sum_{(i,j)\in E_t} \big(x_i(t) - x_j(t)\big)^2$. Assuming that $G_{\leq t_c}$ is connected, Lemmas 9 and 10 imply that

$$\|x\|_q^2 - \|x(t_c+1)\|_q^2 \geq \frac{bh^2}{2d^2} \sum_{t \leq t_c} \sum_{i=1}^{m_t} \Delta_{t,i}^2 \geq \frac{bh^2}{4d^3 mn} \|x - \hat{x}\|_q^2\,. \tag{13}$$

Let $A(n, m)$ be the maximum $s$-energy of an *expS* with at most $n$ vertices and $m$ connected components at any time, subject to the initial condition $\|x - \hat{x}\|_q^2 \leq 1$ and, without loss of generality, $\hat{x} = \mathbf{0}$. By (13), $\|x(t)\|_q^2$ shrinks by at least a factor of $\alpha := 1 - bh^2/(4d^3 mn)$ by time $t_c + 1$. By scaling, we see that the $s$-energy expanded after $t_c$ is at most $\alpha^{s/2} A(n, m)$. While $t < t_c$ (or if $G_{\leq t_c}$ is not connected), the system can be decoupled into two *expS* with fewer than $m$ components. Since $\|x\|_q = 1$, the diameter of the system is at most $2 \max_i |x_i| \leq 2/\sqrt{d}$; therefore $A(n, m) \leq \alpha^{s/2} A(n, m) + 2A(n, m-1) + m\big(2/\sqrt{d}\,\big)^s$. It follows that

$$A(n, m) \leq \frac{2}{1 - \alpha^{s/2}} \Big( A(n, m-1) + m \Big). \tag{14}$$

If $m = 1$ then $t_c = 1$, so we can bypass Lemma 10 and its reliance on the diameter. Instead, we use the connectedness of the graphs to derive from Lemma 9:

$$\|x\|_q^2 - \|x(2)\|_q^2 \geq \frac{1}{d} \sum_{(i,j)\in E_1} (x_i - x_j)^2 \geq \frac{bh^2}{d^3} \|x - \hat{x}\|_q^2\,.$$

Setting $\alpha = 1 - bh^2/d^3$ proves the first part of Theorem 8. The second part follows from (14) and the boundary case $m = 1$ we just derived. ◀

## 2.3 Random averaging systems

It is assumed here that each graph $G_t$ is picked independently, uniformly from the set of simple $(d-1)$-regular graphs with $n$ vertices, with $d > 3$ [58]. (We use $d-1$ because $d$ must account for the self-loops.) The system is a special case of a *revS*, so we use the same notation. The stochastic matrix $P_t$ for $G_t$ is $\frac{1}{d}M_t$, where $M_t$ is a random symmetric 0/1 matrix with a positive diagonal and all row sums equal to $d$; we have $q = d\mathbf{1}$. We define the $s$-energy $\mathcal{E}_s^{\text{RAN}}$ for systems with unit variance $\|x - \hat{x}\,\|_q^2 = 1$, where $\hat{x} = \frac{1}{d}\langle x, \mathbf{1}\rangle$.

▶ **Theorem 11.** *$\mathbb{E}\,\mathcal{E}_s^{\text{RAN}} \le c/s$, for any $s \in (0,1]$, where $c$ is an absolute constant.*

The term absolute refers to the fact that $c$ is independent of the problem's size and parameters; we assume that $d$ is fixed. Let $x_i, y_i$ be the positions of agent $i$ at step $t$ and $t+1$ respectively. As usual, we may place the center of gravity $\hat{x}$ at the origin at time 1, where it will remain forever; that is, $\sum_{i=1}^n x_i = 0$.

▶ **Lemma 12.** *$\mathbb{E}\sum_{i=1}^n y_i^2 = (1-b)\sum_{i=1}^n x_i^2$, where $b = \frac{(d-1)n}{d(n-1)}$.*

**Proof.** Write $\delta_{ij} = x_i - x_j$ and $M_t = (m_{ij})$. With all sums extending from 1 to $n$, we have

$$\sum_i x_i^2 - \sum_i y_i^2 = \sum_i x_i^2 - \sum_i \left(x_i - \frac{1}{d}\sum_j m_{ij}\delta_{ij}\right)^2 = \frac{2}{d}\sum_{i,j} m_{ij}x_i\delta_{ij} - \frac{1}{d^2}\sum_{i,j,k} m_{ij}m_{ik}\delta_{ij}\delta_{ik}$$
$$= \frac{1}{d}\sum_{i,j} m_{ij}\delta_{ij}^2 - \frac{1}{2d^2}\sum_{i,j,k} m_{ij}m_{ik}(\delta_{ij}^2 + \delta_{ik}^2 - \delta_{jk}^2) = \frac{1}{2d^2}\sum_{i,j,k:i\neq j} m_{ik}m_{jk}\delta_{ij}^2\,, \tag{15}$$

with the last equality following from $\sum_{k=1}^n m_{ik} = d$ and $\delta_{ii} = 0$. By symmetry, $\Pr[m_{ij} = 1] = (d-1)/(n-1)$ and $\Pr[m_{ij}m_{ik} = 1] = \binom{d-1}{2}/\binom{n-1}{2}$, for any pairwise distinct $i, j, k$. For any $i \neq j$, we have $\sum_k m_{ik}m_{jk} = 2m_{ij} + \sum_{k:k\neq i,k\neq j} m_{ik}m_{jk}$; hence

$$\sum_k \mathbb{E}\,[m_{ik}m_{jk}] = 2\,\mathbb{E}\,[m_{ij}] + \sum_{k:k\neq i,k\neq j} \mathbb{E}\,[m_{ik}m_{jk}] = \frac{d(d-1)}{n-1}.$$

Since $\sum_i x_i = 0$, we have $\sum_{i,j:i\neq j}\delta_{ij}^2 = 2n\sum_i x_i^2$. By (15), it follows that

$$\mathbb{E}\sum_{i=1}^n y_i^2 = \sum_i x_i^2 - \frac{1}{2d^2}\sum_{i,j,k:i\neq j}\delta_{ij}^2\,\mathbb{E}\,[m_{ik}m_{jk}] \qquad = \sum_i x_i^2 - \frac{d-1}{2d(n-1)}\sum_{i,j:i\neq j}\delta_{ij}^2\,.$$

Markov's inequality tells us that $\sum_i y_i^2 \ge (1-b/3)\sum_i x_i^2$ holds with probability at most $\mathbb{E}\left[\sum_i y_i^2\right] \big/ \left[(1-b/3)\sum_i x_i^2\right] \le 1 - b/2$. Since $\|x\|_q = 1$, the diameter of the system is at most $2/\sqrt{d}$; by the usual scaling law, it follows that

$$\mathbb{E}\,\mathcal{E}_s^{\text{RAN}} \le 2^s\,\mathbb{E}K + \frac{b}{2}\big(1-b/3\big)^{s/2}\,\mathbb{E}\,\mathcal{E}_s^{\text{RAN}} + \big(1-b/2\big)\,\mathbb{E}\,\mathcal{E}_s^{\text{RAN}},$$

where $K$ is the number of connected components in $G_1$. It is known [58] that, for $d > 3$, the probability that the graph is not connected is $O(n^{3-d})$; hence $\mathbb{E}K = O(1)$. Since $b \ge 1/2$, we conclude that $\mathbb{E}\,\mathcal{E}_s^{\text{RAN}} = O\big(1/\big(1-(5/6)^{s/2}\big)\big) = O(1/s)$; hence Theorem 11. ◀

## 3 The Overton Window Attractor

Following in a long line of opinion dynamics models [9, 20, 21, 28, 31], we consider a collection of $n$ agents, each one holding an opinion vector $x_i(t) \in [0,1]^d$ at time $t$; we denote by $x(t)$ the $n$-by-$d$ matrix whose $i$-th row corresponds to $x_i(t)$. Given a stochastic matrix $P_t$, the agents update their opinion vectors at time $t > 0$ according to the evolution equation $x(t+1) = P_t\, x(t)$. We assume that the last $k$ agents $n-k+1, \ldots, n$ are *fixed* in the sense that $x_i(t)$ remains constant at all times $t > 0$. Algebraically, the square block of $P_t$ corresponding to the $k$ fixed agents is set to the identity matrix $\mathbb{I}_k$. The fixed agents can influence the mobile ones, but not the other way around. The presence of fixed agents (also called "stubborn," "forceful" or "zealots" in the literature) has been extensively studied [2, 1, 30, 43, 44, 53, 60, 59].

In the context of social networks, the fixed sources may consist of venues with low user influence, such as news outlets, wiki pages, influencers, TV channels, political campaign sites, etc. [16, 29, 35, 38, 49, 56, 61]. We how how the mobile agents migrate to the convex hull of the fixed agents; crucially, we bound the rate of attraction. This provides both a quantitative illustration of the famous *Overton window* phenomenon as well as a theoretical explanation for why the window acts as an attracting manifold [4, 6, 19, 27, 46]. Interestingly, the emergence of a global attractor does not imply convergence (ie, fixed-point attraction). The mobile agents might still fluctuate widely in perpetuity. The point is that they will always do so within the confines of the global attractor.

To reflect the stochasticity inherent in the choice of sources visited by a user on a given day, we adopt a classic "planted" model: Fix a connected $n$-vertex graph $G$ and two parameters $p \in (0,1]$ and $\rho \in (0, 1/2]$. At each time $t > 0$, $G_t$ is defined by picking every edge of $G$ with probability at least $p$. (No independence is required and $n$ self-loops are included.) We define an $n$-by-$n$ stochastic matrix $P_t$ by setting every entry to 0 and updating it as follows:

1. For $i > n-k$, $(P_t)_{ii} = 1$.
2. For $i \le n-k$, set $(P_t)_{ij} \ge \rho$ for any $j$ such that $(i,j)$ is an edge of $G_t$.

Note that the update is highly nondeterministic. The only two conditions required are that (i) nonzero entries be at least $\rho$ and (ii) each row sum up to 1.

▶ **Theorem 13.** *For any $\delta, \varepsilon > 0$, with probability at least $1-\delta$, all of the agents fall within distance $\varepsilon$ of the convex hull of the fixed agents after a number of steps at most*

$$\frac{1}{p\delta}\left(\frac{c}{\rho}\log\frac{dn}{\varepsilon}\right)^{2(n-1)}$$

*for constant $c > 0$.*

**Proof.** Let $Q_t$ be the $h$-by-$h$ upper-left submatrix of $P_t$, where $h = n-k$. Note that $Q_{\le t} := Q_t \cdots Q_1$ coincides with the $h$-by-$h$ upper-left submatrix $P_{\le t}$. Thus, to show that the mobile agents are attracted to the convex hull of the fixed ones, it suffices to prove that $Q_{\le t}$ tends to $\mathbf{0}_{h\times h}$. To do that, we create an agreement system consisting of $h+1$ agents embedded in $[0,1]$ and evolving as $y(t+1) = A_t\, y(t)$, where: $y(t) \in \mathbb{R}^{h+1}$; $y_{h+1}(1) = 0$; $v = (\mathbb{I}_h - Q_t)\mathbf{1}_h$; and

$$A_t = \begin{pmatrix} Q_t & v \\ \mathbf{0}_h^T & 1 \end{pmatrix}.$$

The system lacks the requisite zero-symmetry to qualify as a *genS*, so we use symmetrization [12] by duplicating the $h$ mobile agents and initializing the embedding of the two copies as mirror-image reflections about the origin. The new evolution matrix is now $\nu$-by-$\nu$, where $\nu = 2h+1$:

$$B_t = \begin{pmatrix} Q_t & v & \mathbf{0}_{h\times h} \\ u & 1-2\|u\|_1 & u \\ \mathbf{0}_{h\times h} & v & Q_t \end{pmatrix}.$$

We define the row vector $u \in \mathbb{R}^h$ by setting its $i$-th coordinate to $\rho$ if $v_i > 0$ and 0 otherwise. We require that $1 - 2\|u\|_1 \geq \rho$; hence $\rho \leq 1/(2d_t + 1)$, where $d_t$ is the number of mobile agents (among the $h$ of them) adjacent in $G_t$ to at least one fixed agent. This condition is easily satisfied by setting $\rho \leq 1/2n$. The evolution follows the update: $z(t+1) = B_t\, z(t)$, where $z(t) \in [-1,1]^\nu$ and $z_{h+1}(1) = 0$.

Let $G^*$ be the augmented $\nu$-vertex graph formed from $G$ and let $G_t^*$ be its subgraph selected at time $t$. Note that, via $z(t)$, these graphs are embedded in $[-1,1]^\nu$. If $\Delta_t$ denotes the length of the longest edge of $G^*$ at time $t$ and $T_\alpha$ is the last time at which the diameter of the system is at least $\alpha$, then $\Delta_t \geq \alpha/\nu$ for all $t \leq T_\alpha$ because $G^*$ is connected. The longest edge in $G^*$ (with ties broken alphabetically) appears in $G_t^*$ with probability at least $p$. Fix $s \in (0,1]$ and define the random variable $\chi_t$ to be $\Delta_t^s$ if the longest edge of $G$ at time $t$ is in $G_t$ and 0 otherwise. By [13], the maximum $s$-energy satisfies $\mathcal{E}_s \leq 2^s(3/\rho s)^{\nu-1}$; hence

$$\mathbb{E}\, T_\alpha \leq \left(\frac{\nu}{\alpha}\right)^s \mathbb{E}\sum_{t\geq 0} \Delta_t^s \leq \frac{1}{p}\left(\frac{\nu}{\alpha}\right)^s \mathbb{E}\sum_{t\geq 0}\chi_t \leq \frac{1}{p}\left(\frac{\nu}{\alpha}\right)^s \mathcal{E}_s \leq \frac{2}{p}\left(\frac{\nu}{\alpha}\right)^s \left(\frac{3}{\rho s}\right)^{\nu-1}.$$

Minimizing the right-hand side over all $s \in (0,1]$ yields

$$\mathbb{E}\, T_\alpha \leq \frac{4}{p}\left(\frac{3}{\rho}\log\frac{2n-1}{\alpha}\right)^{2(n-1)}.$$

By Markov's inequality, $\Pr\left[\, T_\alpha \geq t_\delta \,\right] \leq \delta$, where

$$t_\delta := \frac{4}{p\delta}\left(\frac{3}{\rho}\log\frac{2n-1}{\alpha}\right)^{2(n-1)}. \tag{16}$$

This implies that $\|Q_{\leq t}\, \mathbf{1}_h\|_\infty \leq \alpha$, for all $t > t_\delta$, with probability at least $1-\delta$. In other words, for any such $t$, it holds that, for $i \leq h$,

$$q := \sum_{j=1}^{h}(P_{\leq t})_{ij} = \sum_{j=1}^{h}(Q_{\leq t})_{ij} \leq \alpha.$$

Trivially, $x_i(t+1) = qu + (1-q)v$, where

$$u = \frac{1}{q}\sum_{j=1}^{h}(P_{\leq t})_{ij}x_j(1) \qquad \text{and} \qquad v = \frac{1}{1-q}\sum_{j=h+1}^{n}(P_{\leq t})_{ij}x_j(1).$$

Observing that $v$ lies in the convex hull of the fixed agents, we form the difference $x_i(t+1) - v = q(u-v)$ and note that the distance from $x_i(t+1)$ to the hull is bounded by $q\|u-v\|_2 \leq \alpha\sqrt{d}$. Setting $\alpha = \varepsilon/\sqrt{d}$ completes the proof. ◀

We can extend this result so as to relate convergence to connectivity. We now produce the random graph $G_t$ from fixed connected $G$ as we did above, but if this results in a graph with more than $m$ connected components, we add random edges picked uniformly from $G$ until the number of components drops to $m$. Using Theorem 2 in the proof above leads to a more refined bound:

▶ **Theorem 14.** *For any $\delta, \varepsilon > 0$, with probability at least $1 - \delta$, all of the agents fall within distance $\varepsilon$ of the convex hull of the fixed agents in time bounded by*

$$\frac{1}{p\delta}\left(\frac{1}{\rho}\right)^{2(n-1)}\left(cmn\log\frac{dn}{\varepsilon}\right)^{2m-1},$$

*for constant $c > 0$. This assumes that no graph used in the process has more than $m$ connected components.*

**Proof.** By Theorem 2, we know that $\mathcal{E}^{\text{GEN}}_{m,s} \leq (b/s)^m(1/\rho)^{n-1}$, for any $s \in (0, 1]$, where $b = O(mn)$. The previous proof leads us to update (16) into:

$$t_\delta := \frac{1}{p\delta}\left(\frac{1}{\rho}\right)^{2(n-1)}\left(cmn\log\frac{n}{\alpha}\right)^{2m-1},$$

for constant $c > 0$, from which the theorem follows. ◀

## References


1 Olle Abrahamsson, Danyo Danev, and Erik G. Larsson. Opinion dynamics with random actions and a stubborn agent. In Michael B. Matthews, editor, *53rd Asilomar Conference on Signals, Systems, and Computers, ACSCC 2019, Pacific Grove, CA, USA, November 3-6, 2019*, pages 1486–1490. IEEE, 2019. doi:10.1109/IEEECONF44664.2019.9048901.

2 Daron Acemoglu, Asuman E. Ozdaglar, and Ali ParandehGheibi. Spread of (mis)information in social networks. *Games Econ. Behav.*, 70(2):194–227, 2010. doi:10.1016/j.geb.2010.01.005.

3 Noga Alon. Eigenvalues and expanders. *Comb.*, 6(2):83–96, 1986. doi:10.1007/BF02579166.

4 Emmi Bevensee and Alexander Reid Ross. The alt-right and global information warfare. In Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He, and Jeffrey S. Saltz, editors, *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, pages 4393–4402. IEEE, 2018. doi:10.1109/BigData.2018.8622270.

5 Vincent D. Blondel, Julien M. Hendrickx, and John N. Tsitsiklis. On krause's multi-agent consensus model with state-dependent connectivity. *IEEE Trans. Autom. Control.*, 54(11):2586–2597, 2009. doi:10.1109/TAC.2009.2031211.

6 George-Daniel Bobric. The overton window: A tool for information warfare. In *ICCWS 2021 16th International Conference on Cyber Warfare and Security*, pages 20–27. Academic Conferences Limited, 2021.

7 Francesco Bullo, Jorge Cortés, and Sonia Martinez. *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*, volume 27. Princeton University Press, 2009.

8 Ming Cao, A. Stephen Morse, and Brian D. O. Anderson. Reaching a consensus in a dynamically changing environment: Convergence rates, measurement delays, and asynchronous events. *SIAM J. Control. Optim.*, 47(2):601–623, 2008. doi:10.1137/060657029.

9 Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Reviews of modern physics*, 81(2):591–646, 2009.

10 Bernadette Charron-Bost and Patrick Lambein-Monette. Computing outside the box: Average consensus over dynamic networks. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.10.

11 Samprit Chatterjee and Eugene Seneta. Towards consensus: Some convergence theorems on repeated averaging. *Journal of Applied Probability*, 14(1):89–97, 1977.

12 Bernard Chazelle. The total s-energy of a multiagent system. *SIAM J. Control. Optim.*, 49(4):1680–1706, 2011. doi:10.1137/100791671.

**13** Bernard Chazelle. A sharp bound on the s-energy and its applications to averaging systems. *IEEE Trans. Autom. Control.*, 64(10):4385–4390, 2019. doi:10.1109/TAC.2019.2899509.

**14** Bernard Chazelle. On the periodicity of random walks in dynamic networks. *IEEE Trans. Netw. Sci. Eng.*, 7(3):1337–1343, 2020. doi:10.1109/TNSE.2019.2924921.

**15** Bernard Chazelle and Kritkorn Karntikoon. Quick relaxation in collective motion. In *61st IEEE Conference on Decision and Control, CDC 2022, Cancun, Mexico, December 6-9, 2022*, pages 6472–6477. IEEE, 2022. doi:10.1109/CDC51059.2022.9992475.

**16** Uthsav Chitra and Christopher Musco. Analyzing the impact of filter bubbles on social network polarization. In James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 115–123. ACM, 2020. doi:10.1145/3336191.3371825.

**17** Étienne Coulouma, Emmanuel Godard, and Joseph G. Peters. A characterization of oblivious message adversaries for which consensus is solvable. *Theor. Comput. Sci.*, 584:80–90, 2015. doi:10.1016/j.tcs.2015.01.024.

**18** Felipe Cucker and Steve Smale. Emergent behavior in flocks. *IEEE Trans. Autom. Control.*, 52(5):852–862, 2007. doi:10.1109/TAC.2007.895842.

**19** CIH David Dyjack DrPH. The overton window. *Journal of Environmental Health*, 82(7):54–53, 2020.

**20** Guillaume Deffuant, David Neau, Frédéric Amblard, and Gérard Weisbuch. Mixing beliefs among interacting agents. *Adv. Complex Syst.*, 3(1-4):87–98, 2000. doi:10.1142/S0219525900000078.

**21** Morris H DeGroot. Reaching a consensus. *Journal of the American Statistical association*, 69(345):118–121, 1974.

**22** Matthew G Earl and Steven H Strogatz. Synchronization in oscillator networks with delayed coupling: A stability criterion. *Physical Review E*, 67(3):036204, 2003.

**23** Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. Brief announcement: Broadcasting time in dynamic rooted trees is linear. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25–29, 2022*, pages 54–56. ACM, 2022. doi:10.1145/3519270.3538460.

**24** Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. Asymptotically tight bounds on the time complexity of broadcast and its variants in dynamic networks. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 47:1–47:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.47.

**25** Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. Time complexity of broadcast and consensus for randomized oblivious message adversaries. *CoRR*, abs/2302.11988, 2023. doi:10.48550/arXiv.2302.11988.

**26** Fabio Fagnani and Paolo Frasca. *Introduction to averaging dynamics over networks*. Springer, 2018.

**27** Denis N Fedyanin. On control for reaching a consensus in multiagent systems with bounded opinion updates. In *2021 14th International Conference Management of large-scale system development (MLSD)*, pages 1–5. IEEE, 2021.

**28** Noah E Friedkin and Eugene C Johnsen. Social influence and opinions. *Journal of Mathematical Sociology*, 15(3-4):193–206, 1990.

**29** Jason Gaitonde, Jon M. Kleinberg, and Éva Tardos. Polarization in geometric opinion dynamics. In Péter Biró, Shuchi Chawla, and Federico Echenique, editors, *EC '21: The 22nd ACM Conference on Economics and Computation, Budapest, Hungary, July 18-23, 2021*, pages 499–519. ACM, 2021. doi:10.1145/3465456.3467633.

**30** Javad Ghaderi and R. Srikant. Opinion dynamics in social networks with stubborn agents: Equilibrium and convergence rate. *Autom.*, 50(12):3209–3215, 2014. doi:10.1016/j.automatica.2014.10.034.

31 Rainer Hegselmann and Ulrich Krause. Opinion dynamics and bounded confidence: models, analysis and simulation. *J. Artif. Soc. Soc. Simul.*, 5(3), 2002. URL: http://jasss.soc.surrey.ac.uk/5/3/2.html.

32 Julien Hendrickx and Vincent Blondel. Convergence of different linear and non-linear vicsek models. In *MTNS 2006*, pages 1229–1240, 2006.

33 Julien M. Hendrickx and John N. Tsitsiklis. Convergence of type-symmetric and cut-balanced consensus seeking systems. *IEEE Trans. Autom. Control.*, 58(1):214–218, 2013. doi:10.1109/TAC.2012.2203214.

34 Ali Jadbabaie, Jie Lin, and A. Stephen Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. Autom. Control.*, 48(6):988–1001, 2003. doi:10.1109/TAC.2003.812781.

35 Jing Jiang, Christo Wilson, Xiao Wang, Wenpeng Sha, Peng Huang, Yafei Dai, and Ben Y. Zhao. Understanding latent interactions in online social networks. *ACM Trans. Web*, 7(4):18:1–18:39, 2013. doi:10.1145/2517040.

36 Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 513–522. ACM, 2010. doi:10.1145/1806689.1806760.

37 Fadel Lashhab. *Building Efficiency Control; Dynamic Consensus Networks*. LAP Lambert Academic Publishing, 2018.

38 Kevin Lewis, Marco Gonzalez, and Jason Kaufman. Social selection and peer influence in an online social network. *Proceedings of the National Academy of Sciences*, 109(1):68–72, 2012.

39 Jan Lorenz. A stabilization theorem for dynamics of continuous opinions. *Physica A: Statistical Mechanics and its Applications*, 355(1):217–223, 2005.

40 Jan Lorenz. Heterogeneous bounds of confidence: Meet, discuss and find consensus! *Complex.*, 15(4):43–52, 2010. doi:10.1002/cplx.20295.

41 Anahita Mirtabatabaei and Francesco Bullo. Opinion dynamics in heterogeneous networks: Convergence conjectures and theorems. *SIAM J. Control. Optim.*, 50(5):2763–2785, 2012. doi:10.1137/11082751X.

42 Ciamac Cyrus Moallemi and Benjamin Van Roy. Consensus propagation. *IEEE Transactions on Information Theory*, 52(11):4753–4766, 2006.

43 Mauro Mobilia. Does a single zealot affect an infinite group of voters? *Physical review letters*, 91(2):028701, 2003.

44 Mauro Mobilia, Anna Petersen, and Sidney Redner. On the role of zealotry in the voter model. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(08):P08029, 2007.

45 Luc Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Trans. Autom. Control.*, 50(2):169–182, 2005. doi:10.1109/TAC.2004.841888.

46 Daniel J Morgan. The overton window and a less dogmatic approach to antibiotics. *Clinical Infectious Diseases*, 70(11):2439–2441, 2020.

47 Angelia Nedic, Alex Olshevsky, Asuman E. Ozdaglar, and John N. Tsitsiklis. On distributed averaging algorithms and quantization effects. *IEEE Trans. Autom. Control.*, 54(11):2506–2517, 2009. doi:10.1109/TAC.2009.2031203.

48 Angelia Nedich. Convergence rate of distributed averaging dynamics and optimization in networks. *Found. Trends Syst. Control.*, 2(1):1–100, 2015. doi:10.1561/2600000004.

49 Tien T. Nguyen, Pik-Mai Hui, F. Maxwell Harper, Loren G. Terveen, and Joseph A. Konstan. Exploring the filter bubble: the effect of using recommender systems on content diversity. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 677–686. ACM, 2014. doi:10.1145/2566486.2568012.

50 Alex Olshevsky and John N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM J. Control. Optim.*, 48(1):33–55, 2009. doi:10.1137/060678324.

**51** Alex Olshevsky and John N. Tsitsiklis. Degree fluctuations and the convergence time of consensus algorithms. *IEEE Trans. Autom. Control.*, 58(10):2626–2631, 2013. doi:10.1109/TAC.2013.2257969.

**52** Alistair Sinclair. *Algorithms for random generation and counting – A Markov chain approach.* Progress in theoretical computer science. Birkhäuser, 1993.

**53** Ye Tian and Long Wang. Opinion dynamics in social networks with stubborn agents: An issue-based perspective. *Autom.*, 96:213–223, 2018. doi:10.1016/j.automatica.2018.06.041.

**54** John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

**55** Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, 75(6):1226–1229, 1995.

**56** Bimal Viswanath, Alan Mislove, Meeyoung Cha, and P. Krishna Gummadi. On the evolution of user interaction in facebook. In Jon Crowcroft and Balachander Krishnamurthy, editors, *Proceedings of the 2nd ACM Workshop on Online Social Networks, WOSN 2009, Barcelona, Spain, August 17, 2009*, pages 37–42. ACM, 2009. doi:10.1145/1592665.1592675.

**57** Kyrill Winkler, Ami Paz, Hugo Rincon Galeana, Stefan Schmid, and Ulrich Schmid. The time complexity of consensus under oblivious message adversaries. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 100:1–100:28. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.100.

**58** Nicholas C Wormald et al. Models of random regular graphs. *London Mathematical Society Lecture Note Series*, pages 239–298, 1999.

**59** Ercan Yildiz, Daron Acemoglu, Asuman E Ozdaglar, Amin Saberi, and Anna Scaglione. Discrete opinion dynamics with stubborn agents. *Available at SSRN 1744113*, 2011.

**60** Mehmet Ercan Yildiz, Asuman E. Ozdaglar, Daron Acemoglu, Amin Saberi, and Anna Scaglione. Binary opinion dynamics with stubborn agents. *ACM Trans. Economics and Comput.*, 1(4):19:1–19:30, 2013. doi:10.1145/2538508.

**61** Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social Media Mining: An Introduction.* Cambridge University Press, 2014. doi:10.1017/CBO9781139088510.

# Making Self-Stabilizing Algorithms for Any Locally Greedy Problem

**Johanne Cohen**
Université Paris-Saclay, CNRS, LISN, 91405, Orsay, France

**Laurence Pilard**
LI-PaRAD, UVSQ, Université Paris-Saclay, France

**Mikaël Rabie**
IRIF-CNRS, Université Paris Cité, France

**Jonas Sénizergues**
Université Paris-Saclay, CNRS, LISN, 91405, Orsay, France

## Abstract

Self-stabilizing algorithms are a way to deal with network dynamicity, as it will update itself after a network change (addition or removal of nodes or edges), as long as changes are not frequent. We propose an automatic transformation of synchronous distributed algorithms that solve locally greedy and mendable problems into self-stabilizing algorithms in anonymous networks.

Mendable problems are a generalization of greedy problems where any partial solution may be transformed -instead of completed- into a global solution: every time we extend the partial solution, we are allowed to change the previous partial solution up to a given distance. Locally here means that to extend a solution for a node, we need to look at a constant distance from it.

In order to do this, we propose the first explicit self-stabilizing algorithm computing a $(k, k-1)$-ruling set (*i.e.* a "maximal independent set at distance $k$"). By combining this technique multiple times, we compute a distance-$K$ coloring of the graph. With this coloring we can finally simulate Local model algorithms running in a constant number of rounds, using the colors as unique identifiers.

Our algorithms work under the Gouda daemon, similar to the probabilistic daemon: if an event should eventually happen, it will occur.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms

**Keywords and phrases** Greedy Problem, Ruling Set, Distance-K Coloring, Self-Stabilizing Algorithm

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.11

**Related Version** *Full Version*: https://arxiv.org/abs/2208.14700

## 1 Introduction

The greedy approach is often considered to solve a problem: Is it possible to build up a solution step by step by completing a partial solution? For example, in graph theory, one can consider the Maximal Independent Set (MIS) problem that consists in selecting a set of nodes such that no two chosen nodes are adjacent and any unselected node is a neighbor of a selected one. To produce a MIS, a simple algorithm selects a node, rejects all its neighbors, and then repeats this operation until no node is left. Another classical greedy algorithm is the one that produces a $(\Delta+1)$-coloring of a graph, where $\Delta$ is the maximum degree in the graph. Each time a node is considered, as it has at most $\Delta$ different colors in its neighborhood, one can always choose a different color to extend the current partial solution. Observe that most graphs admit a $\Delta$-coloring, which cannot be found with this heuristic. We can also notice that the size of a MIS can be arbitrarily smaller than the size of a *maximum* independent set. More generally, greedy algorithm are simple algorithm that build not necessarily optimal


© Johanne Cohen, Laurence Pilard, Mikaël Rabie, and Jonas Sénizergues;
licensed under Creative Commons License CC-BY 4.0

solutions. *Greedy problems* are problems that can be solved using a greedy algorithm. We say that a problem is *Locally greedy* if you only need to have information at some constant distance from a node to complete a partial solution on that node. For example $\Delta$-coloring and MIS problems are *Locally greedy* problems while the spanning tree problem is not.

Sometimes, it is not possible to complete a partial solution, and it might be necessary to change some of the outputs to reach a feasible solution. The idea of fixing, or mending a solution, in distributed computing, has been studied a lot, for example in [29, 30]. A formal definition of *Mendable* problems has recently been introduced in [8]. In a graph, we compute the output of each node one after another. For each chosen node, it is possible to change the output of its neighborhood, but only up to some distance. The set of mendable problems is larger than the set of greedy ones. For instance, the 4-coloring of the grid is a mendable problem, but it cannot be solved greedily, as its maximal degree $\Delta$ is equal to 4.

A more generalized way to consider MIS are *ruling sets*. Given a graph $G = (V, E)$, a $(a, b)$-*ruling set* is a subset $S \subset V$ such that the distance between any two nodes in $S$ is at least $a$, and any node in $V$ is at distance at most $b$ from some node in $S$.

In particular, a $(2, 1)$-ruling set is a MIS of $G$. A $(k, k-1)$-ruling set $S$ is a maximal independent set at distance $k$ (also called maximal distance-$k$ independent set): all the elements of $S$ are at distance at least $k$ from each other, every other node is at distance at most $k-1$ from $S$, and thus cannot be added. Note that it is a MIS of $G^{k-1}$ (the graph with the same vertices as $G$, and with edges between two vertices if there are at distance $k-1$ or less from each other in $G$), and this problem can be greedily solved.

A *distance-$K$ coloring* of a graph $G = (V, E)$ is a mapping $\mathcal{C} : V \to \mathbb{N}$ such that for any pair of nodes $u$ $v$ at distance at most $K$ from each other, we have $\mathcal{C}(u) \neq \mathcal{C}(v)$. A way to produce a distance-$K$ coloring is to partition $V$ into sets of nodes at distance at least $k > K$ from each other, *i.e.* distance-$k$ independent sets, each one representing a color. One can construct such a partition sequentially by constructing a partition into $X \geq \Delta^k$ distance-$k$ independent sets $\{S^{(i)}\}_{i \leq X}$, where $S^{(i)}$ is a distance-$k$ independent set of $G$ maximal under the constraint that every node of the independent set must be in $V \setminus \bigcup_{j<i} S^{(j)}$. These distance-$k$ independent sets can be computed similarly to $(k, k-1)$-ruling sets.

The Local model [36] is a synchronous model with unlimited bound on memory where each node starts with a unique identifier. In particular, after $k$ communication rounds, each node knows everything about its neighborhood at distance $k$. A distance-$2K$ coloring allows to simulate Local algorithms running in at most $K$ rounds, as no node can see twice the same identifier in its neighborhood at distance $K$ (see for example [8, 12]).

An algorithm is *Self-stabilizing* if, from any configuration (system state), the system will eventually reach a configuration with a good output/solution (see [3, 17]). In particular, such an algorithm permits one to get back to a good configuration if some faults occur (for example, a node accidentally switches its state). In such situations, being able to locally mend around the fault is key, as it minimises the information and time needed to fix the issue. Self-stabilizing algorithms using mending techniques have been extensively studied, for example in [1, 22, 38]. Self-stabilization can manage the updates in a dynamic network when they occur not too often as adding or removing nodes or links can be viewed as transient faults.

## 1.1 Our Contribution

This paper aims to adapt the idea of Local mending from [8] to produce a self-stabilizing algorithm working in anonymous networks for any constant-radius mendable problem. It uses $f(\Delta) = \Delta^{\Delta^{O(1)}}$ states (in particular, it becomes constant for bounded degree graphs).

In Section 2, we provide a self-stabilizing algorithm that computes a $(k, k-1)$-ruling set in an anonymous network under the Gouda daemon. This algorithm will be used as a sub-routine for our construction. The algorithm detects when a leader can be added (*i.e.* there is a ball of radius $k$ without leader) or two leaders are too close (*i.e.* at distance less than $k$ from each other). To that end, each node computes its distance from the leaders. If a node and its neighbors are at distance at least $k-1$ from the leaders, that node can try to add itself to the ruling set. If two leaders are too close, thanks to a clock system consisting of a mosaic of local synchronizers beta of Awerbuch [4], a node in the middle of the path will eventually detect the problem and initiate the removal of the leaders from the set. Thanks to the Gouda daemon, we ensure that only a few nodes will try to add themselves simultaneously and that the clock system will eventually detect collisions. Section 3 contains the proof that a stable configuration can always be reached, and the Gouda daemon ensures that it ultimately happens.

In Section 4, by combining this algorithm $\Delta^k$ times, we partition the graph into distance-$k$ independent sets, which corresponds to a distance-$K$ coloring for any $K < k$. This coloring allows us to consider nodes of each set sequentially to compute a solution to some greedy problem. In Section 5, we present a solution allowing us to solve any $T$-mendable problem in anonymous networks, where $T$ is a constant corresponding to the radius up to which we are permitted to change the output of a node. To that end, we use the fact that a Local algorithm runs in $r$ rounds for some constant $r$, when a distance-$2T+1$ coloring is given. To do that, we compute a distance-$2T+1$ and a distance-$2r+1$ coloring. That way, each node can access their neighborhood at the proper distance and compute the output the Local algorithm would have given in that situation.

## 1.2 Related Work

The notion of *checking locally* was introduced by Afek et al. [2] and its relationship with the idea of *solving locally* by Naor and Stockmeyer [31]. This work, along with Cole and Vishkin's algorithm that efficiently computes a 3-coloring of a ring [14], leads to the notion of *Locally Checkable Labelling problems* (Lcl) and the Local model. Locally checkable problems are problems such that when the output is locally correct for each node, the global output is guaranteed to be correct too. Coloring and MIS belong to that field. Ruling Sets are also Lcl problems: to check locally that the solution is correct, the distance to the set must be given in the output. The Local model (see [36] for a survey) is a synchronous model that requires unique identifiers but does not impose any restriction on communication bandwidth or computation complexity. The goal is to find sublinear time algorithms. An adaptation of the Local model, the Slocal model [21] considers algorithms executed on nodes one after another, only one time each, but are allowed to see the state of every node up to some distance when they do. In particular, this model solves locally greedy problems with a constant distance of sight.

Bitton et al. [11] designed a self-stabilizing transformer for Local problems. Their probabilistic transformer converts a given fault-free synchronous algorithm for Lcl problems into a self-stabilizing synchronous algorithm for the same problem in anonymous networks. The overheads of this transformation in terms of message complexity and average time complexity are upper bounded: the produced algorithms stabilize in time proportional to $\log(\alpha + \Delta)$ in expectation, where $\alpha$ is the number of faulty nodes. Afek and Dolev [1] designed a self-stabilizing transformer. It converts any distributed algorithm that works in a network with identifiers and diameter less than $D$ under the synchronous daemon into a self-stabilizing one adding additional costs in time (additional $O(D)$), memory ($O(nD)$ multiplier), and communication ($O(nD)$ multiplier).

Awerbuch et al. [5] introduced the ruling set as a tool for decomposing the graph into small-diameter connected components. As for the seminal work, the ruling set problems have been used as a sub-routine function to solve some other distributed problems (network decompositions [5, 10], colorings [32], shortest paths [27]).

The MIS problem has been extensively studied in the LOCAL model, [19, 34, 13] for instance and in the CONGEST model [33] (synchronous model where messages are $O(\log n)$ bits long). In the LOCAL model, Barenboim et al. [9] focused on systems with unique identifiers and gave a self-stabilizing algorithm producing an MIS within $O(\Delta+\log^* n)$ rounds. Balliu et al. [6] prove that the previous algorithm [9] is optimal for a wide range of parameters in the LOCAL model. In the CONGEST model, Ghaffari et al. [20] prove that there exists a randomized distributed algorithm that computes a maximal independent set in $O(\log \Delta \cdot \log\log n + \log^6 \log n)$ rounds with high probability. Considering the problem $(\alpha, \beta)$-ruling set in a more general way, Balliu et al. [7] give some lower bound for computing a $(2, \beta)$- ruling set in the LOCAL model: any deterministic algorithm requires $\Omega\left(\min\left\{\frac{\log \Delta}{\beta \log\log \Delta}, \log n\right\}\right)$ rounds.

Up to our knowledge, no self-stabilizing algorithm has been designed for only computing $(k, k-1)$-ruling sets where $k > 2$ under the Gouda daemon. Self-stabilizing algorithms for maximal independent set have been designed in various models (anonymous network [35, 40, 39] or not [23, 28, 37]). Shukla et al. [35] present the first self-stabilization algorithm for finding a MIS for anonymous networks. Turau [37] gives the best-known result with $O(n)$ moves under the distributed daemon. Recently, some works improved the results in the synchronous model. For non-anonymous networks, Hedetniemi [26] designed a self-stabilization algorithm that stabilizes in $O(n)$ synchronous rounds. Moreover, for anonymous networks, Turau [39] designs some randomized self-stabilizing algorithms for maximal independent set that stabilizes in $O(\log n)$ rounds w.h.p. See the survey [25] for more details on MIS self-stabilizing algorithms.

Our algorithm uses a clock system close to information propagation with feedback (or PIF) mechanism, however more than these classical solutions are needed. Indeed, while we assume multiple leaders, in classical PIF algorithms, only one leader is usually assumed under identified system [15] or anonymous one [16]. Their mechanism relies on waves of information from a source to the network, layer by layer.

## 1.3 Model

A distributed system consists of a set of processes where two adjacent processes can communicate. The communication relation is represented by a graph $G = (V, E)$ where $V$ is the set of the processes (we call *node* any element of $V$ from now on) and $E$ represents the neighborhood relation between them, *i.e.*, $uv \in E$ when $u$ and $v$ are adjacent nodes. The set of *neighbors* of a node $u$ is denoted by $N(u)$. We assume the system to be *anonymous*, meaning that a node has no identifier. Moreover, we consider undirected networks (i.e. $uv \in E \iff vu \in E$). We denote by $\Delta$ the maximum degree in the graph.

We denote by $dist(u, v)$ the distance between the two nodes $u$ and $v$ in the graph. When $S$ is a subset of $V$, $dist(u, S)$ is the smallest distance from $u$ to an element in $S$. In what follows, the concept of *ball* will play an important role. Formally, the *ball* of radius $i$ and center $s$, $\mathcal{B}(s, i)$, is the set of nodes that are at distance at most $i$ from $s$. Observe that a ball of radius $a-1$ centered in a node of the ruling set $S$ contains only one node in $S$.

For communication, we consider the *shared memory model*: the *local state* of each node corresponds to a set of *local variables*. A node can read its local variables and its neighbors' but can only rewrite its local variables. A *configuration* is the value of the local states of all nodes in the system. When $u$ is a node and $x$ is a local variable, the *$x$-value* of $u$ is the

value $x_u$. Each node executes the same algorithm that consists of a set of *rules*. Each rule is of the form *"**if** ⟨guard⟩ **then** ⟨command⟩"* and is parameterized by the node where it would be applied. Each rule also has a priority number. The *guard* is a predicate over the variables of the current node and its neighbors. The *command* is a sequence of actions that may change the values of the node's variables (but not those of its neighbors). A rule is *activable* in a configuration $C$ if its guard in $C$ is true. A process is *eligible* for the rule $\mathcal{R}$ in a configuration $C$ if its rule $\mathcal{R}$ is activable and no rule of lower priority number is activable for that node in $C$. We say in that case that the process is *activable* in $C$. An *execution* is an alternate sequence of configurations and actions $\sigma = C_0, A_0, \ldots, C_i, A_i, \ldots$, such that $\forall i \in \mathbb{N}^*$, configuration $C_{i+1}$ is obtained by executing the command of at least one rule that is activable in configuration $C_i$. More precisely, the set of actions $A_i$ is the non-empty set of activable processes in $C_i$ such that their activable rules have been executed to reach $C_{i+1}$.

The goal of a self-stabilizing algorithm is to be robust to perturbations. An initial configuration cannot follow any restriction, and failures can occur, changing the state of some of the nodes. A self-stabilizing algorithm must be able to recover and reach a correct general output from any configuration.

In a distributed system, multiple nodes can be active simultaneously, meaning they are in a state where they can make a computation. The definition of a self-stabilizing algorithm is centred around the notion of *daemon*. A *daemon* captures which set of activable rules some scheduler choose during the execution. See [18] for a taxonomy. Our algorithm cannot work on a fully synchronous deterministic anonymous network, as it relies on using asynchronous clocks from different leaders. To that end, we use the *Gouda* daemon to break symmetries, as it ensures asynchronous activation of the nodes. We aim to create algorithms for any mendable problems to solve the computability question. Hence, we do not focus on the complexity time, which could be captured by a probabilistic daemon. The Gouda daemon captures the same computable problems as the probabilistic daemon. If something happens with probability 1 with the probabilistic daemon (where each rule has a probability $< 1$ to be activated), it eventually happens with the Gouda daemon.

▶ **Definition 1** ([18, 24]). *We say that an execution $\sigma = C_0 \to C_1 \to C_2 \ldots$ is under the Gouda daemon if: for any configurations $C$ and $C'$ such that $C \to C'$ can be executed, if $C$ appears infinitely often in $\sigma$, then $C'$ also appears infinitely often in $\sigma$.*

An algorithm is *self-stabilizing* for a given specification (i.e. a set of restrictions over the configurations) under some daemon if there exists a subset $\mathcal{L}$ of the set of all configurations, called the *legitimate configurations*, such that: (i) any configuration in $\mathcal{L}$ verifies the specification, and any execution under the said daemon starting in $\mathcal{L}$ stays in $\mathcal{L}$ (*correctness*). and (ii) any execution under the said daemon eventually reaches a configuration in $\mathcal{L}$ (*convergence*). The set $\mathcal{L}$ is called the set of *legitimate configurations*.

## 2 Self-Stabilizing Algorithm for Computing a $(k, k-1)$-Ruling Set

### 2.1 General Overview

As we want to compute a $(k, k-1)$-ruling set, a node needs to detect when it is currently "too far" from the nodes pretending to be in the ruling set. When $k = 2$, a $(2, 1)$-ruling set is an MIS, and some self-stabilization algorithms are designed for finding an MIS [35, 40, 39]. For the remaining of the document, we assume $k > 2$.

To this aim, the local variable $d$ represents the distance at which the node thinks it is from the ruling set. In particular, a $d$-value of 0 indicates that a node is (or thinks it is) in the ruling set, and we denote by $S(C)$ the set of those nodes in a given configuration $C$.

**Algorithm 1** Algorithm for the $(k, k-1)$-Ruling Set.

——— **Attributes of the nodes**

$\mathbf{d_u \in [\![0, k-1]\!]}$
$\mathbf{err_u \in \{0,1\}}$
For every $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!]$ : $\mathbf{c_{i,u} \in \mathbb{Z}/4\mathbb{Z}}$ and $\mathbf{b_{i,u} \in \{\uparrow, \downarrow\}}$

——— **Predicates**

well_defined$(u) \equiv err_u = 0 \wedge \forall v \in N(u), |d_u - d_v| \leq 1 \wedge (d_u > 0 \Rightarrow (\exists v \in N(u), d_v = d_u - 1))$
leader_down$(u) \equiv d_u = 0 \Rightarrow \forall i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \downarrow$
branch_coherence_up$(u, i) \equiv$
$\forall v \in N(u), d_v = d_u - 1 \Rightarrow (b_{i,u}, b_{i,v}, c_{i,v}) \in \{(\uparrow, \uparrow, c_{i,u}), (\uparrow, \downarrow, c_{i,u}), (\uparrow, \downarrow, c_{i,u}+1), (\downarrow, \downarrow, c_{i,u})\}$
branch_coherence_down$(u, i) \equiv$
$\forall v \in N(u), d_v = d_u + 1 \Rightarrow (b_{i,u}, b_{i,v}, c_{i,v}) \in \{(\uparrow, \uparrow, c_{i,u}), (\downarrow, \uparrow, c_{i,u}), (\downarrow, \uparrow, c_{i,u}-1), (\downarrow, \downarrow, c_{i,u})\}$
branch_coherence$(u) \equiv d_u \geq \lfloor \frac{k}{2} \rfloor \vee \big(branch_coherence_up(u, d_u) \wedge$
$\forall i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!], branch\_coherence\_up(u, i) \wedge branch\_coherence\_down(u, i)\big)$

——— **Rules**

**Incr Leader::** (priority 2)
**if** well_defined$(u) \wedge (d_u = 0) \wedge (\exists i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], \forall v \in N(u), d_v = 1 \wedge c_{i,u} - c_{i,v} = 0)$
**then** For all such $i$, $c_{i,u} := c_{i,u} + 1$

**Sync 1 down::** (priority 2)
**if** well_defined$(u) \wedge \exists! v \in N(u), \exists i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], d_u = 1 \wedge d_v = 0 \wedge c_{i,u} = c_{i,v} - 1 \wedge b_{i,u} = \uparrow)$
**then** For all such $i$, $c_{i,u} := c_{i,v}$ ; $b_{i,u} := \downarrow$

**Sync 2+ down::** (priority 2)
**if** well_defined$(u) \wedge 1 < d_u < \lfloor \frac{k}{2} \rfloor$
$\wedge(\exists i \in [\![d_u, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \uparrow \wedge \forall v \in N(u), d_v = d_u - 1 \Rightarrow (c_{i,u} = c_{i,v} - 1 \wedge b_{i,v} = \downarrow))$
**then** For all such $i$, $c_{i,u} := c_{i,v}$ ; $b_{i,u} := \downarrow$

**Sync 1+ up::** (priority 2)
**if** well_defined$(u) \wedge 0 < d_u < \lfloor \frac{k}{2} \rfloor$
$\wedge(\exists i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \downarrow \wedge \forall v \in N(u), d_v = d_u + 1 \Rightarrow (c_{i,u} = c_{i,v} \wedge b_{i,v} = \uparrow))$
**then** For all such $i$, $b_{i,u} := \uparrow$

**Sync end-of-chain::** (priority 2)
**if** well_defined$(u) \wedge 0 < d_u < \lfloor \frac{k}{2} \rfloor \wedge \forall v \in N(u), d_v = d_u - 1 \Rightarrow (c_{d_u,u} = c_{d_u,v} - 1 \wedge b_{i,v} = \downarrow))$
**then** $b_{d_u,u} := \uparrow$ ; $c_{d_u,u} := c_{i,v}$

**Update distance ::** (priority 0)
**if** $(d_u \neq 0) \wedge d_u \neq \min(\min\{d_v | v \in N(u)\} + 1, k-1)$
**then** $d_u := \min(\min\{d_v | v \in N(u)\} + 1, k-1)$
If $d_u < \lfloor \frac{k}{2} \rfloor$ : Let $v := \text{choose}(\{w \in N(u) | d_w = d_u - 1\})$
For each $i \in [\![d_u, \lfloor \frac{k}{2} \rfloor - 1]\!], c_{i,u} := c_{i,v}$ ; $b_{i,u} := b_{i,v}$

**Become Leader ::** (priority 2)
**if** $err_u = 0 \wedge (d_u = k-1) \wedge \forall v \in N(u), d_v = k-1$
**then** $d_u := 0$, For each $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], c_{i,u} := 0, b_{i,u} := \downarrow$

**Leader down ::** (priority 1)
**if** $well\_defined(u) \wedge d_u = 0 \wedge \exists i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \uparrow$ **then** For each $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!]$, $b_{i,u} := \downarrow$

**Two Heads::** (priority 1)
**if** $err_u = 0 \wedge \exists v, v' \in (N(u) \cup \{u\})^2, v \neq v' \wedge d_v = d_{v'} = 0)$ **then** $err_u := 1$

**Branch incoherence::** (priority 1)
**if** $err_u = 0 \wedge \neg branch\_coherence(u)$ **then** $err_u := 1$

**Error Spread ::** (priority 2)
**if** $err_u = 0 \wedge (d_u \leq \lfloor \frac{k}{2} \rfloor - 1) \wedge (\exists v \in N(u), err_v = 1 \wedge d_u < d_v)$ **then** $err_u := 1$

**Reset Error ::** (priority 2)
**if** $(err_u = 1) \wedge ([d_u > \lfloor \frac{k}{2} \rfloor] \vee [\forall v \in N(u), d_v \geq d_u \vee err_v = 1])$
**then** $err_u := 0$, If $d_u = 0$, $d_u := 1$, For each $i$, $c_{i,u} := 0$, $b_{i,u} := \uparrow$

Any other value of $d_u$ represents the distance to $S(C)$ (the minimum between $k-1$ and the said distance). The rule **Update distance** has the highest priority. Its goal is to ensure that each node eventually gets its distance to $S(C)$ accurately. When a node $u$ has its local variable $d_u$ equal to $k-1$ and is surrounded by nodes of $d$-value $k-1$, it "knows" that it is far enough from $S(C)$ to be added to it. Node $u$ can then execute rule **Become Leader** to do so. Update of $d$-values will then spread from the new member of $S(C)$ through the execution of rule **Update distance**.

The way to insert new nodes into $S(C)$ cannot avoid the fact that two new members of $S(C)$ may be too close. A way to detect those problems is needed to guarantee that we will not let those nodes in $S(C)$.

If they are close enough (distance 2 or less), it can be directly detected by a node (either a common neighbor if they are at distance 2 or one of them if they are at distance 1). The rule **Two Heads** is here to detect this.

No node can detect this problem when problematic nodes are too far away. To remedy this, each node maintains a synchronized clock system around each node of $S(C)$ by executing the *stationary rules*. For this reason, we split the set of rules into two groups:

- The *stationary* rules are the rules **Incr Leader**, **Sync 1 down**, **Sync 2+ down**, **Sync 1+ up**, and **Sync end-of-chain**;
- The *convergence* rules are the rules **Remote Collision**, **Two Heads**, **Branch Incoherence**, **Update Distance**, **Become Leader**, **Error Spread**, **Reset Error**, and **Leader down**.

We say that a node in $S(C)$ is the *leader* of the nodes under its influence, corresponding to the nodes in its ball at distance $\lfloor \frac{k}{2} \rfloor$. Assuming $d$-value has already been spread, the clock of index $i$ of nodes that gave the same leader will always be either equal or out-of-sync by 1. Thus, a node detects that two nodes in $S(C)$ are too close when it sees in its neighbourhood two nodes with clocks out-of-sync by 2. It will raise an error when activated by executing rule **Remote Collision**. The error is then propagated toward the problematic members of $S(C)$ by rule **Error Spread**.

In both previous cases, the problematic nodes of $S(C)$ end up having $err$-value 1, which makes them leave $S(C)$ by executing rule **Reset Error**. Afterwards, rule **Update distance** will, over time, update the $d$-values of the nodes at distance up to $k$ to that node.

The goal of our algorithm is to ensure that we reach locally a configuration from which, when a node is inserted in $S(C)$, and no node gets added at distance at most $k-1$ away, it remains in $S(C)$ forever. Note that when it is executed, rule **Update distance** setup the clock values and arrows (variables $c$ and $b$) so that the newly updated node is synchronized to its "parent" (the node it takes as a reference to update its $d$-value).

The target configuration is **not** a *stable* configuration, and from it, all the nodes can only execute *stationary rules*. In this configuration, $S(C)$ is guaranteed to be a $(k, k-1)$-ruling set of the underlying graph. Note that the predicate $well\_defined$ appears in the guard of every stationary rule. The predicate guarantees that the considered node neither is in error-detection mode nor has some incorrect $d$-values in its neighbourhood before executing any clock-related rule.

## 2.2 The Clock System

Now, we describe the clock system that detects two leader nodes in $S(C)$ at a distance less than $k$. The leaders are the nodes that update the clock value $c_i$ and propagate it to its "children" and so on. For a given clock index $i$, when every neighbour of a leader $s$ has the same clock $c_i$ and their corresponding arrow $b_i$ pointed up, node $s$ increments its clock value by 1 by executing rule **Incr Leader**.

After that, the clock value is propagated downward (toward nodes of greater $d$-value) using rules **Sync 1 down** and **Sync 2+ down**. Note that it is performed locally by layers: one node of a given $d$-value cannot update its clock value and arrow before every neighbour with a smaller $d$-value does so. This is necessary to guarantee the global synchronization of the clock.

There are two ways for the propagation of $(c_i, b_i)$ to reach the limit of the area it should spread in: either it has reached nodes with $d$-value $i$, or there is no node having a greater $d$-value to spread the clock further.

- In the first case, rule **Sync end-of-chain** flips the arrow $b_i$.
- In the second case, the nodes execute rule **Sync 1+ up** to flip $b_i$.

In both cases, it allows rule **Sync 1+ up** to propagate upward (toward smaller $d$-values) with the $b_i$-value switching to $\uparrow$ from the nodes to their parents. Note that it is done locally by layers: one node of a given $d$-value may not update its clock value and arrow before every neighbour with a greater $d$-value has done so.

When the propagation reaches the neighbors of $s$, node $s$ "detects" that its current clock value has been successfully propagated, and it will execute rule **Incr Leader** to increase it.

The point of this clock system is that two nodes under the same leader cannot have clock values out-of-sync by 2, but two nodes that have different leaders may. It allows them to detect a "collision" (*i.e.* two nodes of $S(C)$ too close from each other) when the $d$-values of two such nodes are smaller than $\lfloor \frac{k}{2} \rfloor$. Observe that the clock of index $i$ is only reliable for detecting collision between nodes of $S(C)$ at distance $2i$ or $2i+1$ from each other. For smaller distances, this clock may be forcefully synchronized between two nodes of $S(C)$ by layer-by-layer updating, and for greater distances, no node may detect an out-of-sync from it. This process differs from the PIF mechanism [15]: we need to run one clock for each layer, as a clock of higher layer will be synchronized for the two conflicting leaders because of further nodes. Thus, we have $\lfloor \frac{k}{2} \rfloor - 1$ parallel clock systems to capture every possible distance of collision.

The Gouda daemon ensures that if two nodes of $S(C)$ are too close, this will only be the case for a while. The clock system will eventually detect it and propagate an error.

### 2.3 Handling Initial and Perturbed Configurations

Rules **Leader Down** and **Branch Incoherence** are only executed to solve problems coming from the initial configuration or after a perturbation has occurred. Rule **Leader Down** is executed when a leader has some of its arrows $b_i$ in the wrong direction. Rule **Branch Incoherence** is executed when some "impossible" patterns are produced in the clock systems due to wrong clock values and arrows in the initial state. Standard patterns are shown in Figure 1. Any other pattern will make an activated node to execute rule **Branch Incoherence**.

## 3 Proof of the Algorithm

### 3.1 Stability of Legitimate Configurations

The ruling set algorithm presented in this section uses the state model. It constructs the set of vertices whose $d$-value is 0. We will prove that this set is a ruling set in legitimate configurations. Formally, we require the following specification for the legitimate configurations:

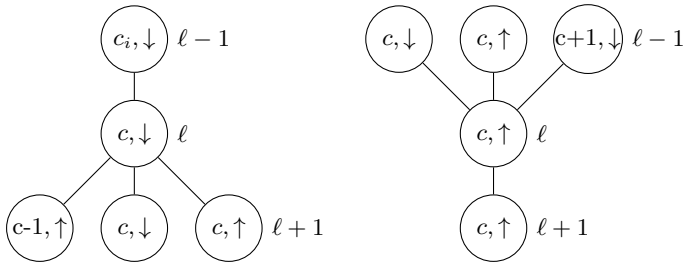



**Figure 1** Branch coherence condition. The couples $(c, \downarrow)$ or $(c, \uparrow)$ represent the local variables $(c_i, b_i)$ of the nodes. The value on the right of the node represents its distance to the leader node (i.e. its $d$-value). The central node in both figures is the reference, and the other nodes represent the possible couples for its neighbors with different $d$-value.

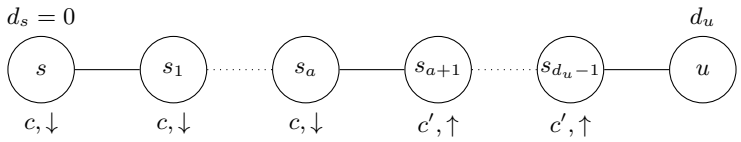



**Figure 2** Node $s$ propagates its clock value along a shortest path from $s$ to $u$ where $c' \in \{c, c-1\}$.

▶ **Definition 2.** *Let $S(C)$ be the set of nodes $s$ such that $d_s = 0$ in a given configuration $C$. Configuration $C$ is said to be legitimate if:*

1. *for any $u$ we have well_defined$(u)$, leader_down$(u)$ and branch_coherence$(u)$ hold;*
2. *for any two distinct nodes $u$ and $v$ of $S(C)$, we have $dist(u, v) \geq k$.*

▶ **Theorem 3.** *The set of legitimate configurations is closed. Moreover, all the $d$-values do not change from a legitimate configuration $C$.*

Thanks to Theorem 3, we know that, from a legitimate configuration, we keep the same set of leaders $S(C)$, which forms a $(k, k-1)$-ruling set. Hence, under the Gouda daemon, the set of leaders will eventually be a stable $(k, k-1)$-ruling set.

The goal of the following lemmas will be to prove Theorem 3. Lemma 4 ensures that $S(C)$ forms a ruling set when the values of all the local variables are correct.

▶ **Lemma 4.** *Let $C$ be a legitimate configuration. For any node $u$, $d_u = dist(u, S(C))$, and $S(C)$ is a $(k, k-1)$-ruling set of the underlying graph.*

Now we focus on the clock system. We prove the following property on the ruling set to run the clock system.

▶ **Lemma 5.** *Let $C$ be a legitimate configuration and $s$ be a node in $S(C)$. For every node $u$, $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor$ implies that $d_u = dist(u, s)$.*

This property allows us to deduce that a node $u$ such that $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor$ has only one node $s$ of $S(C)$ in its ball at distance $\lfloor \frac{k}{2} \rfloor$. Thus, all the nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ must be synchronized with $s$. We explain how the values representing the clock of the local variable of nodes with $d$-value smaller than $\lfloor \frac{k}{2} \rfloor$ are spread from their leader. Figure 2 illustrates how the pairs $(c_i, b_i)$ go from nodes in $S(C)$.

▶ **Lemma 6.** *Let $C$ be a legitimate configuration and $s$ a node in $S(C)$. For every node $u$ such that $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor - 1$, every shortest path $(s_0, s_1, \cdots, s_{d_u})$ from $s$ to $u$ satisfies the following property in $C$:*

*For every clock index* $i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!]$*, there exists some integer* $a \in [\![0, d_u]\!]$ *such that:*
1. $\forall \ell \in [\![0, a]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\downarrow, c_{i,s})$;
2. $\exists c' \in \{c_{i,s} - 1, c_{i,s}\}, \forall \ell \in [\![a + 1, d_u]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\uparrow, c')$.

Lemma 7 proves that only rules to update clocks are executed from legitimate configurations:

▶ **Lemma 7.** *Let* $C$ *be a legitimate configuration. Let* $u$ *be a node. Node* $u$ *only executes stationary rules from* $C$*.*

Once the execution reaches a legitimate configuration $C$, we have proved that only stationary rules can be executed. The goal is to use that result and the previous lemmas to prove that only legitimate configurations can be reached from $C$. This result will lead to the proof of Theorem 3.

## 3.2 Reaching a Legitimate Configuration

The goal of the following lemmas is to prove that, from any configuration $C$, we can reach a configuration $C'$ that is legitimate. The Gouda daemon's property concludes that a legitimate configuration will always eventually be reached. Indeed, let $C$ be a configuration that is infinitely often reached during an execution. Under the Gouda daemon, as a legitimate configuration $C'$ is reachable from configuration $C$, $C'$ will also be reached infinitely often.

To that end, we introduce the notion of *locally legitimate* node for leaders satisfying conditions close to the legitimate ones in their ball of radius $k - 1$. We prove that if a node $s$ is locally legitimate, then it will remain so forever (Lemma 11).

We explain how to make locally legitimate a node with no leader at a distance smaller than $k$ to it in Lemmas 13 and 16. We explain how, when some leaders are too close to each other, we can reach a configuration where none of the remaining ones are at distance smaller than $k$ from another (Lemma 16).

From here, we can conclude with the proof of the following theorem:

▶ **Theorem 8.** *Under the Gouda daemon, any execution eventually reaches a legitimate configuration.*

We first introduce the notion we will use in this section for nodes in $S(C)$:

▶ **Definition 9.** *Let* $C$ *be a configuration. A node* $s$ *in* $S(C)$ *is locally legitimate if*
1. *all the nodes* $u$ *in* $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ *are such that* $well\_defined(u)$*,* $leader\_down(u)$ *and* $branch\_coherence(u)$ *hold and* $d_u = dist(u, s)$*;*
2. *all the nodes* $u$ *in* $\mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ *are such that* $k - dist(u, s) \leq d_u \leq dist(u, s)$*.*

*We denote* $\mathcal{LL}(C)$ *the set of those nodes in* $C$*.*

Let $s$ be a locally legitimate node. The first property means that in its neighbourhood at distance at most $\lfloor \frac{k}{2} \rfloor$, nodes behave like in a legitimate configuration. Therefore, they cannot detect errors. The second property implies that all nodes in $\mathcal{B}(s, k-1)$ have coherent $d$-values according to $s$ and to potential leaders at distance at least $k$ from $s$. A direct observation is the following:

▶ **Lemma 10.** *Let* $s \in \mathcal{LL}(C)$*. We have* $\mathcal{B}(s, k-1) \cap S(C) = \{s\}$*.*

Combining Lemma 10 and the first property of the legitimated node, we can deduce that once a node is legitimate, it remains legitimate during the rest of the execution.

▶ **Lemma 11.** *Let* $C$*,* $C'$ *be two configurations such that* $C \to C'$*. We have* $\mathcal{LL}(C) \subset \mathcal{LL}(C')$*.*

We focus now on how to create locally legitimate nodes. First of all, we can make sure that the $d$-values of all the nodes are coherent with regards to their distance to $S(C)$:

▶ **Lemma 12.** *For any configuration $C$, we can reach a configuration $C'$ such that $S(C) = S(C')$, and $d_u = \min(dist(u, S(C')), k-1)$ for every node $u$, and there is no node with err-value 1 among nodes with d-value greater than $\lfloor \frac{k}{2} \rfloor$.*

Let $s$ be a node at distance at least $k$ from $S(C)$. We explain how to make that node locally legitimate:

▶ **Lemma 13.** *Let $C$ be a configuration where there exists a node $s$ such that $dist(s, S(C)) \geq k$. A configuration $C'$ can be reached from $C$ such that $s \in \mathcal{LL}(C')$.*

Now, we need to deal with leaders that are too close from each other. To do this, we introduce the function that measures the number of nodes in this situation in a configuration, and Lemma 15 shows how to decrease it.

▶ **Definition 14.** *Let $C$ be a configuration. We define $\phi(C)$ as the set of leaders in $C$ having a conflict with another one due to being at distance less than $k$ to each other, i.e. $\phi(C) = \{u \in S(C) \mid \exists v \in S(C) \setminus \{u\}, dist(u, v) < k\}$.*

▶ **Lemma 15.** *Let $C$ be a configuration such that $\phi(C) \neq \emptyset$. There exists a node $u$ in $\phi(C)$ and a configuration $C'$ such that we can reach $C'$ from $C$ with $S(C') = S(C) \setminus \{u\}$.*

Thanks to this result, we prove that we can reach a configuration $C$ such that the set of conflicting nodes is empty:

▶ **Lemma 16.** *From any configuration $C$, we can reach a configuration $C'$ such that $\phi(C') = 0$.*

Now we focus on how to make leaders locally legitimate if they do not have any other leaders at distance smaller than $k$ from them.

▶ **Lemma 17.** *Let $C$ and $s$ be a configuration and a node such that $\mathcal{B}(s, k-1) \cap S(C) = \{s\}$. We can reach a configuration $C'$ such that $s \in \mathcal{LL}(C')$.*

Now, we can prove that the number of legitimate nodes increases during the execution up until we converge to a legitimate configuration:

▶ **Lemma 18.** *Let $C$ be a configuration. From $C$, we can reach a configuration $C'$ such that either $\mathcal{LL}(C) \subsetneq \mathcal{LL}(C')$ or $C'$ is legitimate.*

This last lemma allows us to conclude with the proof of Theorem 8.

## 4 From Ruling Sets to Distance-$K$ Colorings

In this section, we focus on the *distance-$K$ coloring* problem. A distance-$K$ coloring is a coloring such that any pair of nodes cannot share a color unless they are at distance greater than $K$. If the nodes having the same color form a $(K+1, K)$-ruling set, then those nodes respect the coloring constraint.

Let choose $k > K$ for our $(k, k-1)$-ruling sets. We partition the set of nodes into two-by-two disjoint sets $S^{(i)}$ such that each set corresponds to nodes of the same color. We build these sets one after another. Each of these sets is a distance-$k$ independent set of the graph, which is maximal among the nodes of $V \setminus \bigcup_{j<i} S^{(j)}(C)$. These sets will be built by composing an adaptation of our $(k, k-1)$-ruling set algorithm. Since the maximum degree

of the graph is $\Delta$, any ball of radius $k-1$ contains at most $\Delta^{k-1}+1$ nodes. Hence we can partition the nodes into $\Delta^k$ ruling sets (we use this majoration in order to simplify the reading of the following proofs).

For this reason, the distance $K$-coloring algorithm is composed of $\Delta^k$ parallel algorithms, each one of them computing an adapted $(k, k-1)$-ruling set. For Algorithm $i$ and configuration $C$, we note $S^{(i)}(C)$ (or $S^{(i)}$ if there is no ambiguity) the corresponding set $S(C)$. Each time a node $u$ is active, it applies a rule (if it can) for each ruling set algorithm.

It is necessary to ensure that a node belongs to exactly one ruling set. To perform this, we number the ruling set algorithms: we denote by $d_u^{(j)}$ the local variable $d_u$ of $u$ of the $j$-th algorithm. By convention, we assume that $u$ belongs to the $j$-th ruling set (or it has color $j$) if $j = min\{1 \leq p \leq \Delta^k \mid d_u^{(p)} = 0\}$. To form a partition with the sets, we need to reach a configuration where for each node $u$, $|\{i \leq \Delta^k \mid d_u^{(i)} = 0\}| = 1$. To achieve this, we modify rule **Become Leader** and add a rule to detect if a node is a leader in different layers (for Algorithm $j$).

**Become Leader$^{(j)}$ ::** (priority 1)
**if** $err_u^{(j)} = 0 \wedge (d_u^{(j)} = k-1) \wedge \forall v \in N(u), d_v^{(j)} = k-1 \wedge \forall p < j : d_u^{(p)} > 0$
**then** $d_u^{(j)} := 0$
$\forall i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], c_{i,u}^{(j)} := 0, b_{i,u}^{(j)} :=\downarrow$

**Belong To Two ruling sets$^{(j)}$ ::** (priority 0)
**if** $d_u^{(j)} = 0 \wedge \exists p < j : d_u^{(p)} = 0$
**then** $d_u^{(j)} := 1$

We also modify the predicate $well\_defined$ (for Algorithm $j$) as follows, which impacts the definition of legitimate configuration. In particular, now, a node $u$ such that $d_u^{(j)} = k-1$ does not need to have a neighbor closer to a leader if $d_u^{(i)} = 0$ for some $i < j$.

$\text{well\_defined}^{(j)}(u) \equiv err_u^{(j)} = 0 \wedge \forall v \in N(u), |d_u^{(j)} - d_v^{(j)}| \leq 1 \wedge$
$((\forall p \leq j, d_u^{(p)} > 0) \vee d_u^{(j)} < k-1 \Rightarrow (\exists v \in N(u), d_v^{(j)} = d_u^{(j)} - 1)) \wedge (d_u^{(j)} = 0 \Rightarrow \forall p < j, d_u^{(p)} > 0)$

We give a new definition of *legitimate configuration*:

▶ **Definition 19.** *Let $j \leq \Delta^k$. A configuration $C$ is said to be legitimate for Algorithm $j$ if, for all $i \leq j$:*

1. *for any $u$ we have $well\_defined^{(i)}(u)$, $leader\_down^{(i)}(u)$ and $branch\_coherence^{(i)}(u)$ hold;*
2. *for any $u \neq v$ in $S^{(i)}(C)^2$, we have $dist^{(i)}(u,v) \geq k$.*

From this, we get the following adaptation of Lemma 4. The proof remains slightly the same, with the exception that in the case of $d_u^{(j)} = k-1$, only nodes that have not a variable $d_u^{(i)} = 0$ for some $i < j$ are considered.

▶ **Lemma 20.** *Let $C$ be a legitimate configuration for Algorithm $j$.*

- *For any node $u$, if for all $i < j$, $d_u^{(i)} > 0$, we have $d_u^{(j)} = dist(u, S^{(j)}(C))$;*
- *For any node $u$, if $d_u^{(i)} = 0$, for all $j > i$, we have $d_u^{(j)} = \min(dist(u, S^{(j)}(C)), k-1)$;*
- *$S^{(j)}(C)$ is a $(k, k-1)$-ruling set of $V \setminus \bigcup_{i<j} S^{(i)}(C)$.*

With these modifications, we have the following adaptation of Theorem 3:

▶ **Theorem 21.** *For all $j \leq \Delta^k$, the set of legitimate configurations for Algorithm $j$ is closed. Moreover, from a legitimate configuration $C$ for Algorithm $j$, all the $d^{(j)}$-value do not change.*

The proof to reach a legitimate configuration for Algorithm $\Delta^k$ works in the same way as the proof of Theorem 8. We need to do it one algorithm after another, from 1 to $\Delta^k$. The main difference is that we only consider nodes that are not a leader in a smaller algorithm when we increase the set of locally legitimate nodes. This leads to the result:

▶ **Theorem 22.** *Under the Gouda daemon, any execution eventually reaches a legitimate configuration in Algorithm $\Delta^k$.*

These two theorems lead to the main result of distance-$K$ coloring:

▶ **Theorem 23.** *Let $k$ and $K$ be two integers such that $k > K$. Under the Gouda daemon, any execution eventually reaches a configuration $C$ such that*

- *$S^{(i)}(C) = \{u : d_u^{(i)} = 0\}$ forms a distance-$k$ MIS of $V \setminus \bigcup_{j<i} S^{(j)}(C)$ in $G$*
- *The sets $S^{(1)}(C), \dots S^{(\Delta^k)}(C)$ form a distance-$K$ coloring.*
- *Every configuration in any execution starting in $C$ verifies the two above properties with the same sets as $C$.*

## 5 Solving Mendable Problems

In this section, we want to solve a generalisation of *Greedy Problems*: $O(1)$-*Mendable Problems*, introduced in [8]. Greedy problems, such as $\Delta + 1$-coloring and Maximal Independent Set, have the property that if some of the nodes have chosen an output that is locally valid (no pair of neighbors sharing a color, no adjacent nodes selected in the set), then any single node can choose an output that will keep the global solution locally valid. In a distributed setting, we cannot do this process sequentially from one node to another, but we can do it in parallel: if a set of nodes that are far enough from each other choose their output at each step, the solution can be completed. The global solution is valid if we repeat this process until all nodes have chosen an output. To that end, we first introduce some definitions.

### 5.1 Definitions

We call a *Locally Checkable Problem* (LCL) $\Pi$ a problem where each node can check locally that its output is compatible with its neighbours. Let $\mathcal{O}$ be the set of outputs. The output $\Gamma : V \to \mathcal{O}$ is good if and only if, for all $u \in V$, $\Gamma(u)$ is compatible with the multiset $\{\Gamma(v) \mid v \in N(u)\}$. For example, in the case of Maximal Independent Set, with $\mathcal{O} = \{0, 1\}$, 1 is compatible with $\{0^k \mid k \leq \Delta\}$, and 0 is compatible with $\{11^x0^y \mid x + y < \Delta\}$. Note that we can consider radius-$r$ neighbourhood for the compatibility in the general case, which we will not do here out of simplicity. Our results can be adapted to the general version.

Let $\mathcal{O}$ be the set of outputs, and $\Gamma^* : V \to \mathcal{O} \cup \{\bot\}$. We say that $\Gamma^*$ is a partial solution if, for any $u \in V$ such that $\Gamma^*(u) \neq \bot$, we can complete the labels of the neighbors $v$ of $u$ (i.e. give an output to the nodes $v$ such that $\Gamma^*(v) = \bot$) to make $u$ compatible with it neighbors.

A problem is $T$-*mendable* if, from any partial solution $\Gamma^*$ and any $v \in V$ such that $\Gamma^*(v) = \bot$, there exists a partial solution $\Gamma'$ such that $\Gamma'(v) \neq \bot$, $\forall u \neq v$, and $\Gamma'(u) = \bot \Leftrightarrow \Gamma^*(u) = \bot$, and $\forall u \in V$, $dist(u, v) > T \Rightarrow \Gamma'(u) = \Gamma^*(u)$. Intuitively, we can change the output of nodes at distance at most $T$ from a node $v$ when we select the output of $v$.

The LOCAL *model* is a synchronous model where each node is given a unique identifier. As there is no limit on the size of the messages for communication, after $r$ rounds, each node knows the topology of their neighborhood at distance $r$.

▶ **Theorem 24** (Restated Theorem 6.2 from [8])**.** *Let $\Pi$ be a $T$-mendable* LCL *problem. $\Pi$ can be solved in $O(T\Delta^{2T})$ rounds in the* LOCAL *model if we are given a distance-$2T + 1$ coloring.*

One can observe that unicity of identifiers provided by the LOCAL model is not necessary to solve an LCL problem as long as nodes do not see twice the same identifier in the run. If we know that an algorithm runs on a graph of size at most $n$ in $r(n) = o(\log n)$ rounds, then we can have it run on any graph of size at least $n$ with a distance-$r(n)$ coloring, using those colors as the new identifiers. The algorithm will not notice that the identifiers are not unique, producing a correct output. This technique has been used, for example, in [8, 12].

Hence, for a constant $T$, we can produce a distance-$r(T)$ coloring to then use the algorithm of Theorem 24.

## 5.2 Solving Greedy and Mendable Problems

The goal now is to use distance-$k$ colorings to solve other problems. Let us say we want to solve $K$-mendable problem $\Pi$ for which we already have a LOCAL algorithm $\mathcal{A}$ from Theorem 24 (the output of node $u$ will be denoted $out_u$). To that end, we first build $\mathcal{A}'$, a self-stabilizing version of $\mathcal{A}$ that solves $\Pi$ assuming unique identifiers at distance $r$. Then we compose $\mathcal{A}'$ with our distance $k$-coloring algorithm (for $k$ big enough) - described in Section 4 - and obtain then a self-stabilizing anonymous algorithm solving $\Pi$. To simulate $r$ rounds in the LOCAL model, we need to compute the graph's topology at distance $r$ for each node. To compute the output of node $u$, $\mathcal{A}'$ will compute the exact mapping of the ball of radius $r$ centered on $u$. From it, $\mathcal{A}'$ will provide the output $\mathcal{A}$ would produce on this ball if the colors were identifiers.

In the following, we describe how each node will compute its ball. If we have beforehand a distance-$2r+1$ coloring, each node will have at most one node of some given color in its neighborhood at distance $r$. Hence, each node can compute a mapping of its neighborhood at distance $r$. At the beginning, each node knows its mapping at distance 0. If all the neighbors of a node $u$ know their mapping at distance $i$, $u$ can deduce its topology up to distance $i+1$. Note that we consider only cases where $r$ does not depend on the size of the graph.

▶ **Lemma 25.** *Let $C$ be a configuration where each node $u$ has a color $c_u$ corresponding to a distance-$2r+1$ coloring and outputs $out_u = \perp$. From this configuration, under the Gouda daemon, we will reach a configuration $C'$ where each node outputs a mapping of their neighborhood at distance $r$.*

With this lemma and Theorem 24, we can conclude to the end result of this section:

▶ **Theorem 26.** *Let $\Pi$ be an LCL problem with mending radius $k$, that can be solved in $r = O(k\Delta^{2k})$ rounds in the LOCAL model. Let $C$ be a configuration where each node $u$ has a color $c_u$ corresponding to a distance-$2k+1$ coloring, a color $c'_u$ corresponding to a distance-$2r+1$ coloring, and outputs $out_u = \perp$. From this configuration, under the Gouda daemon, we will reach a configuration $C'$ where each node outputs a solution to $\Pi$.*

Note that in a ball of radius $2r+1$ in a graph of maximal degree $\Delta$, there are at most $\Delta^{2r+1}$ nodes. Hence, we need $\Delta^{2r+1}$ colors. For graphs where $\Delta$ is constant, we get a constant number of colors. As we also consider constant radius $r$ for the mendability, there are a finite number of possible mappings of balls at distance $r$ using those colors. Hence, in that case, our algorithms use a finite memory that does not depend on the size of the graph.

# 6 Conclusion

This work provides a self-stabilizing algorithm under the Gouda daemon for any locally mendable problem by first introducing an explicit algorithm to compute a $(k, k-1)$-ruling set. This construction generalises well to probabilistic daemons if stationary rules and rule

**Become Leader** have some probability smaller than 1 to be activated. This algorithm permits building up distance-$k$ colorings, which helps solve greedy and mendable problems by simulating the LOCAL model. In the case of constant bounded degree $\Delta$, our algorithm uses a constant memory. We did not consider complexity questions. Considering a probabilistic daemon, an open question would be what complexities can be aimed, as our algorithm did not optimize this question at all.

The presented algorithm for the ruling set should adapt well in the Byzantine case, as the influence of a Byzantine node is naturally confined by the algorithm. In such context, Distance-$K$ identifiers computed in Section 4 would be unique at distance $K$ for nodes far enough from Byzantine nodes. It should be of interest to investigate this point.

## References


1 Yehuda Afek and Shlomi Dolev. Local stabilizer. *Journal of Parallel and Distributed Computing*, 62(5):745–765, 2002.

2 Yehuda Afek, Shay Kutten, and Moti Yung. Local detection for global self stabilization. *Theoretical Computer Science*, 186(1-2):339, 1991.

3 Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. Introduction to distributed self-stabilizing algorithms. *Synthesis Lectures on Distributed Computing Theory*, 8(1):1–165, 2019.

4 Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.

5 Baruch Awerbuch, Andrew V Goldberg, Michael Luby, and Serge A Plotkin. Network decomposition and locality in distributed computation. In *FOCS*, volume 30, pages 364–369. Citeseer, 1989.

6 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *Journal of the ACM (JACM)*, 68(5):1–30, 2021.

7 Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed lower bounds for ruling sets. *SIAM Journal on Computing*, 51(1):70–115, 2022.

8 Alkida Balliu, Juho Hirvonen, Darya Melnyk, Dennis Olivetti, Joel Rybicki, and Jukka Suomela. Local mending. In *International Colloquium on Structural Information and Communication Complexity*, pages 1–20. Springer, 2022.

9 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed ($\delta+$ 1) -coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 437–446, 2018.

10 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)*, 63(3):1–45, 2016.

11 Shimon Bitton, Yuval Emek, Taisuke Izumi, and Shay Kutten. Fully adaptive self-stabilizing transformer for lcl problems. *arXiv preprint*, 2021. `arXiv:2105.09756`.

12 Sebastian Brandt, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Patric RJ Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. Lcl problems on grids. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 101–110, 2017.

13 Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Computing*, 33(3):349–366, 2020.

14 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

15 Alain Cournier, AK Datta, Franck Petit, and Vincent Villain. Self-stabilizing pif algorithm in arbitrary rooted networks. In *Proceedings 21st International Conference on Distributed Computing Systems*, pages 91–98. IEEE, 2001.

16 Alain Cournier, Stéphane Devismes, and Vincent Villain. Snap-stabilizing pif and useless computations. In *12th International Conference on Parallel and Distributed Systems-(ICPADS'06)*, volume 1, pages 8–pp. IEEE, 2006.

17 Shlomi Dolev. *Self-stabilization*. MIT press, 2000.

18 Swan Dubois and Sébastien Tixeuil. A taxonomy of daemons in self-stabilization. *arXiv preprint*, 2011. `arXiv:1110.0334`.

19 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 270–277. SIAM, 2016.

20 Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Improved deterministic network decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2904–2923. SIAM, 2021.

21 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 784–797, 2017.

22 Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 45–54, 1996.

23 Wayne Goddard, Stephen T Hedetniemi, David Pokrass Jacobs, and Pradip K Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 14–pp. IEEE, 2003.

24 Mohamed G Gouda. The theory of weak stabilization. In *International Workshop on Self-Stabilizing Systems*, pages 114–123. Springer, 2001.

25 Nabil Guellati and Hamamache Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *Journal of Parallel and Distributed Computing*, 70(4):406–415, 2010.

26 Stephen T Hedetniemi. Self-stabilizing domination algorithms. *Structures of Domination in Graphs*, pages 485–520, 2021.

27 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. *SIAM Journal on Computing*, 50(3):STOC16–98, 2019.

28 Michiyo Ikeda, Sayaka Kamei, and Hirotsugu Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. In *the Third International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 70–74. Citeseer, 2002.

29 Shay Kutten and David Peleg. Fault-local distributed mending. *Journal of Algorithms*, 30(1):144–165, 1999.

30 Shay Kutten and David Peleg. Tight fault locality. *SIAM Journal on Computing*, 30(1):247–268, 2000.

31 Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

32 Alessandro Panconesi and Aravind Srinivasan. The local nature of $\delta$-coloring and its algorithmic applications. *Combinatorica*, 15(2):255–280, 1995.

33 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

34 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 350–363, 2020.

35 Sandeep K Shukla, Daniel J Rosenkrantz, S Sekharipuram Ravi, et al. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the second workshop on self-stabilizing systems*, volume 7, page 15, 1995.

**36** Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 45(2):1–40, 2013.

**37** Volker Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Information Processing Letters*, 103(3):88–93, 2007.

**38** Volker Turau. Computing fault-containment times of self-stabilizing algorithms using lumped markov chains. *Algorithms*, 11(5):58, 2018.

**39** Volker Turau. Making randomized algorithms self-stabilizing. In *International Colloquium on Structural Information and Communication Complexity*, pages 309–324. Springer, 2019.

**40** Volker Turau and Christoph Weyer. Randomized self-stabilizing algorithms for wireless sensor networks. In *Self-Organizing Systems*, pages 74–89. Springer, 2006.

# Covert Computation in the Abstract Tile-Assembly Model

**Robert M. Alaniz**
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

**David Caballero**
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

**Timothy Gomez**
Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA

**Elise Grizzell**
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

**Andrew Rodriguez**
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

**Robert Schweller**
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

**Tim Wylie**
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

## Abstract

There have been many advances in molecular computation that offer benefits such as targeted drug delivery, nanoscale mapping, and improved classification of nanoscale organisms. This power led to recent work exploring privacy in the computation, specifically, covert computation in self-assembling circuits. Here, we prove several important results related to the concept of a hidden computation in the most well-known model of self-assembly, the Abstract Tile-Assembly Model (aTAM). We show that in 2D, surprisingly, the model is capable of covert computation, but only with an exponential-sized assembly. We also show that the model is capable of covert computation with polynomial-sized assemblies with only one step in the third dimension (just-barely 3D). Finally, we investigate types of functions that can be covertly computed as members of P/Poly.

**2012 ACM Subject Classification** Theory of computation → Computational complexity and cryptography

**Keywords and phrases** self-assembly, covert computation, atam

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.12

**Funding** This research was supported in part by National Science Foundation Grant CCF-1817602.

## 1 Introduction

With the ability to manufacture nanoscale structures and to use DNA as building blocks for structures [28] or for data storage [10], there has been a great increase in the need to process and compute information at the same level. Thus, the study of self-assembling computation has been an important and active area of research over the last two decades.

Designing self-assembling systems that compute functions is an active and well-studied area of computational geometry and biology [4,19]. This ability to craft monomers capable of placing themselves – especially when doing precision construction and computation at scales where conventional tools are incapable of operating, e.g., the nanoscale – has tremendous power. One of the few downsides to self-assembly computation is that the entire history of the computation is visible. In certain cases, this may be undesirable for privacy or security reasons, which we motivate below. Thus, we build on recent work [6,7,9] to explore *covert computation*, where we build Tile Assembly Computers (TACs) designed with the goal of

© Robert M. Alaniz, David Caballero, Timothy Gomez, Elise Grizzell, Andrew Rodriguez, Robert Schweller, and Tim Wylie;
licensed under Creative Commons License CC-BY 4.0



obtaining the output of computation while obscuring the inputs and computational history. We do this by proving that covert computation is possible even in one of the simplest standard models of self-assembly: the Abstract Tile-Assembly Model (aTAM) [29].

**Motivation.** The development of covert computation as a model and method of designing self-assembling systems was driven by several areas of concern in cryptography, biomedical engineering, privacy, and might even help protect intellectual property in systems that use "products of nature," such as DNA, as they cannot be patented in the United States as of 2013 [14]. Covert computation has also emerged as a powerful complexity tool, being used to show the coNP-completeness of the *Unique Assembly Verification* problem in the negative glue aTAM [9], and the PSPACE-completeness of the Unique Assembly Verification problem in the Staged Assembly Model [6]. As this paper focuses on systems without detachment, there might also be important applications in implantable systems where even the possibility of displacement from free-floating DNA could cause unknown side effects or destabilization of the assembly [23].

## 1.1 Previous Work

The Abstract Tile-Assembly Model (aTAM) was first introduced in [29] and inherited the ability to perform Turing computation from Wang tiles. Since then, investigation into the model has led in many directions, such as Intrinsic Universality [18, 22], efficient assembly of shapes [25], and parallel computation [5, 24]. Many generalizations have also appeared, such as allowing for RNA tiles that can be deleted [1, 15], multiple stages of growth [6, 12, 16], and even negative glues [9, 17]. The aTAM is powerful because not only can the tile set store information, but work has also gone into using the seed [3], or even the temperature [11, 26], for making systems more complex.

Tile Assembly Computers were defined in [5, 24], and Covert Computation, as defined in the field of self-assembly, was first introduced in 2019 [9] for negative growth-only aTAM. In *negative* variations of tile self-assembly models, tiles are capable of not only attachment to but also detachment from an assembly if the remaining assembly is still stable. In negative *growth-only* aTAM, tiles are never allowed to detach even though there may be glues providing a repellent force, and the system must be designed so that detachment does not occur. This paper introduced the covert construction framework to answer an open complexity problem for Unique Assembly Verification (UAV) with negative growth-only glues in the aTAM model, showing it to be coNP-complete. Notably, without negative glues, the UAV problem is solvable in polynomial time [2].

Covert computation has been explored in two other models of self-assembly as well: Staged Self-Assembly [6] and Tile Automata [7]. The staged self-assembly model, one of the most powerful *passive* tile self-assembly models, abstracts the process of scientists mixing test tubes together by allowing multiple self-assembly processes occurring in separate "bins" that may be combined in subsequent "stages". The authors show that 3-stages suffice for covert computation and used the techniques to show that the UAV problem directly relates the number of stages to a specific level of the polynomial hierarchy. Thus, with no restrictions on the number of stages, UAV in the staged model is PSPACE-complete. Covert Computation in the *active* self-assembly model of Tile Automata was shown to be rather simple as tiles in the model are capable of changing states (instead of having static glues), easily erasing computational history.

**Table 1** Known Covert Circuits for $n$-bit function $f(x)$. Let MCS be the minimum circuit size that computes $f(x)$. **Input Size** is the size of the input assembly. **Output Size** is the size of the output template, where we use $k$ to describe the number of output bits. * currently only works for binary functions.

| Class | Model | Size Of | | | | Ref |
|---|---|---|---|---|---|---|
| | | Input | Tile Set | Output | Assembly | |
| Bool. Circuits | $\text{Neg}_{\text{GO}}$ | $\mathcal{O}(n)$ | $\mathcal{O}(MCS)$ | $\mathcal{O}(k)$ | $\mathcal{O}(MCS)$ | [9] |
| Bool. Circuits | 3D | $\mathcal{O}(n)$ | $\mathcal{O}(MCS)$ | $\mathcal{O}(k)$ | $\mathcal{O}(MCS)$ | Thm. 1 |
| Rev. Circuits* | 2D | $\mathcal{O}(n+MSC)$ | $\mathcal{O}(MCS)$ | $\mathcal{O}(1)$ | $\mathcal{O}(2^n)$ | Thm. 2 |

## 1.2 Our Contributions

In this work, we further explore the problem of designing covert tile assembly computers (TACs) in the aTAM, focusing on TACs that have a polynomial size description. We provide two new covert computers in the aTAM with only positive glue strengths of $\{1, 2\}$ in Sections 3 and 4. The 3D construction uses a similar technique to the circuits in [9] by implementing a NAND gate using dual rail logic and backfilling. We refer to this covert TAC as having a strict polynomial size since the systems defined by the TAC all produce assemblies of polynomial size. This only uses a single-step into the third dimension, which is occasionally referred to as *just-barely* 3D [20, 21].

The covert TAC in Section 4 is in the standard 2D aTAM. The TAC is of polynomial size, but produces an exponential-size terminal assembly. This works by computing the function non-covertly using Toffoli gates, getting the output, reversing the computation to recover the input, then building the next and previous circuit assemblies until all possible circuits are built. We utilize the Toffoli gates' reversibility property to have a symmetrical circuit assembly that displays its input on both sides that we can increment or decrement (the input used) to start the next computation.

In Section 5 we explore the classes of decision problems solvable by polynomial size covert TACs. Table 1 gives an overview of known covert circuits for functions based on the input size. Since covert has been defined as a non-uniform model, meaning different input sizes have different tile sets, we look at non-uniform complexity classes as well. Namely, the class P/poly, the class of problems solvable by polynomial size circuits. We prove that if a problem is solvable by a 3D covert TAC, then it is in P/poly. This, taken with the result in Section 3, shows an equivalence between these two models of computation.

# 2 Definitions

We begin with an overview of the Abstract Tile-Assembly Model, then follow with a definition of Tile Assembly Computers and covert computation.

## 2.1 Abstract Tile Assembly Model

At a high level, the Abstract Tile-Assembly Model (aTAM) uses a set of *tiles* capable of sticking together to construct shapes. These tiles are typically squares (2D) or cubes (3D) with *glues* on each side where they may attach to one another. A glue is labeled to indicate its type, governing what other tiles it may bond with and the *strength* of the bond. A tile with all of its labels is a *tile type*. A *tile set* contains all the tile types of the system. A single tile may attach at a location if the combined strength of the matching glues is greater than

or equal to the *temperature* $\tau$. An *assembly* is a shape made up of one or more combined tiles. Construction is started around a designated *seed* assembly $S$. Any assembly capable of being made from the seed is called a *producible* assembly. An assembly is *terminal* if no more tiles can attach. A terminal assembly is said to be *uniquely produced* if it is the only terminal assembly that can be made by a tile system. A tile system is formally represented as an ordered triplet $\Gamma = (T, s, \tau)$ of the tile set, seed assembly, and temperature parameter, respectively.

### 2.1.1 aTAM Formal Definitions

**Tiles.** Let $\Pi$ be an alphabet of symbols called the *glue types*. A tile is a finite edge polygon with some finite subset of border points, each assigned a glue type from $\Pi$. Each glue type $g \in \Pi$ also has some integer strength $str(g)$. Here, we consider unit square tiles of the same orientation with at most one glue type per face, and the *location* to be the center of the tile located at integer coordinates.

**Assemblies.** An assembly $A$ is a finite set of tiles whose interiors do not overlap. If each tile in $A$ is a translation of some tile in a set of tiles $T$, we say that $A$ is an assembly over tile set $T$. For a given assembly $A$, define the *bond graph* $G_A$ to be the weighted graph in which each element of $A$ is a vertex, and the weight of an edge between two tiles is the strength of the overlapping matching glue points between the two tiles. Only overlapping glues of the same type contribute a non-zero weight, whereas overlapping, non-equal glues contribute zero weight to the bond graph. The property that only equal glue types interact with each other is referred to as the *diagonal glue function* property, and is perhaps more feasible than more general glue functions for experimental implementation (see [13] for the theoretical impact of relaxing this constraint). An assembly $A$ is said to be $\tau$*-stable* for an integer $\tau$ if the min-cut of $G_A$ has weight at least $\tau$.

**Tile Attachment.** Given a tile $t$, an integer $\tau$, and an assembly $A$, we say that $t$ may attach to $A$ at temperature $\tau$ to form $A'$ if there exists a translation $t'$ of $t$ such that $A' = A \cup \{t'\}$, and the sum of newly bonded glues between $t'$ and $A$ meets or exceeds $\tau$. For a tile set $T$, we use notation $A \rightarrow_{T,\tau} A'$ to denote there exists some $t \in T$ that may attach to $A$ to form $A'$ at temperature $\tau$. When $T$ and $\tau$ are implied, we simply say $A \rightarrow A'$. Further, we say that $A \rightarrow^* A'$ if either $A = A'$, or there exists a finite sequence of assemblies $\langle A_1 \dots A_k \rangle$ such that $A \rightarrow A_1 \rightarrow \dots \rightarrow A_k \rightarrow A'$.

**Tile Systems.** A tile system $\Gamma = (T, S, \tau)$ is an ordered triplet consisting of a set of tiles $T$ called the system's *tile set*, a $\tau$-stable assembly $S$ called the system's *seed* assembly, and a positive integer $\tau$ referred to as the system's *temperature*. A tile system $\Gamma = (T, S, \tau)$ has an associated set of *producible* assemblies, $\texttt{PROD}_\Gamma$, which define what assemblies can grow from the initial seed $S$ by any sequence of temperature $\tau$ tile attachments from $T$. Formally, $S \in \texttt{PROD}_\Gamma$ is a base case producible assembly. Further, for every $A \in \texttt{PROD}_\Gamma$, if $A \rightarrow_{T,\tau} A'$, then $A' \in \texttt{PROD}_\Gamma$. That is, assembly $S$ is producible, and for every producible assembly $A$, if $A$ can grow into $A'$, then $A'$ is also producible.

We further denote a producible assembly $A$ to be *terminal* if $A$ has no attachable tile from $T$ at temperature $\tau$. We say a system $\Gamma = (T, S, \tau)$ *uniquely produces* an assembly $A$ if all producible assemblies can grow into $A$ through some sequence of tile attachments. More formally, $\Gamma$ *uniquely produces* an assembly $A \in \texttt{PROD}_\Gamma$ if for every $A' \in \texttt{PROD}_\Gamma$ it is the case that $A' \rightarrow^* A$. Systems that uniquely produce one assembly are said to be *deterministic*.

## 2.2 Covert Computation

Here, we provide formal definitions for computing a function with a tile system and the further requirements for the covert computation of a function. Our formulation of computing functions is that used in [9], which is a modified version of the definition provided in [24] to allow for each bit to be represented by a subassembly potentially larger than a single tile.

**Tile Assembly Computers (TAC).** Informally, a Tile Assembly Computer (TAC) for a function $f$ consists of a set of tiles, along with a format for both input and output. The input format is a specification for how to build an input seed to the system that encodes the desired input bit-string for function $f$. We require that each bit of the input be mapped to one of two assemblies for the respective bit position: a sub-assembly representing "0" or a sub-assembly representing "1". The input seed for the entire string is the union of all these sub-assemblies. This seed, along with the tile set of the TAC, forms a tile system. The output of the computation is the final terminal assembly this system builds. To interpret what bit-string is represented by the output, a second *output* format specifies a pair of sub-assemblies for each bit. The bit-string represented by the union of these subassemblies within the constructed assembly is the output of the system.

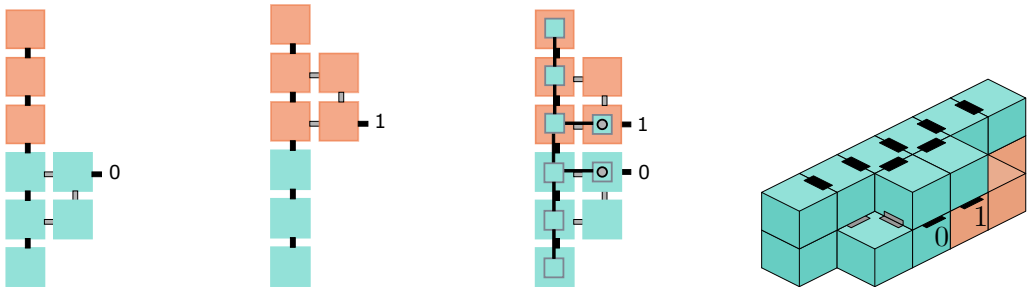
For a TAC to *covertly* compute $f$, the TAC must compute $f$ and produce a unique assembly for each possible output of $f$. We note that our formulation for providing input and interpreting output is quite rigid and may prohibit more exotic forms of computation. Further, we caution that any formulation must take care to prevent "cheating" that could allow the output of a function to be partially or completely encoded within the input. To prevent this, a type of *uniformity* constraint, akin to what is considered in circuit complexity [27], should be enforced. We now provide the formal definitions of function computing and covert computation.

**Input/Output Templates.** An $n$-bit input/output template over tile set $T$ is a sequence of ordered pairs of assemblies over $T$: $A = (A_{0,0}, A_{0,1}), \ldots, (A_{n-1,0}, A_{n-1,1})$. For a given $n$-bit string $b = b_0, \ldots, b_{n-1}$ and $n$-bit input/output template $A$, the *representation* of $b$ with respect to $A$ is the assembly $A(b) = \bigcup_i A_{i,b_i}$. A template is valid for a temperature $\tau$ if this union never contains overlaps for any choice of $b$ and is always $\tau$-stable. An assembly $B \supseteq A(b)$, which contains $A(b)$ as a subassembly, is said to represent $b$ as long as $A(d) \nsubseteq B$ for any $d \neq b$. We refer to the size of a template as the size of the largest assembly defined by the template.

**Function Computing Problem.** A *tile assembly computer* (TAC) is an ordered quadruple $\Im = (T, I, O, \tau)$ where $T$ is a tile set, $I$ is an $n$-bit input template, and $O$ is a $k$-bit output template. A TAC is said to compute function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^k$ if for any $b \in \mathbb{Z}_2^n$ and $c \in \mathbb{Z}_2^k$ such that $f(b) = c$, then the tile system $\Gamma_{\Im,b} = (T, I(b), \tau)$ uniquely assembles a set of assemblies which all represent $c$ with respect to template $O$.

**Covert Computation.** A TAC *covertly* computes a function $f(b) = c$ if 1) it computes $f$, and 2) for each $c$, there exists a unique assembly $A_c$ such that for all $b$, where $f(b) = c$, the system $\Gamma_{\Im,b} = (T, I(b), \tau)$ uniquely produces $A_c$. In other words, $A_c$ is determined by $c$, and every $b$ where $f(b) = c$ has the exact same final assembly.

**Polynomial-Sized Tile Assembly Computers.** We say a TAC is polynomial size if the input template, tile set, and output template are all polynomial in $n$. However, this requirement still allows the producible assemblies to be exponentially larger. We say a TAC is *strictly* polynomial size if the produced assemblies are also polynomial in size.




**(a)** Bit Assembly Template 0. **(b)** Bit Assembly Template 1. **(c)** Bit Assembly Output Template. **(d)** Bit Assembly Output Template (isometric view).

**Figure 1** Input assemblies and their respective input templates. The blue squares represent the bit set to zero, and the orange squares represent a bit set to one. Grey glues are strength-1, black glues are strength-2.

## 3 3-Dimensional Covert Circuits

In this section, we show how to perform covert computation in the aTAM using 3 dimensions. The computation behaves similarly to the covert circuit construction in [9] by building NAND gates and FANOUTs using dual rail logic. We start with showing a NOT that switches which wire is "on", then extending to a NAND by utilizing cooperative binding.

The main difference between the two constructions is when *backfilling* occurs, which is the process of filling in the unused dual rail line once that line is no longer needed. Here, we do not backfill as we go, rather, we fill in the assembly once the computation is complete.

### 3.1 Input Assemblies

Our input assembly consists of $n$ $1 \times 6$ columns with two of four tiles attached on the right (Figures 1a and 1b). The top two tiles will be included when the input is 1, and the bottom two tiles if the input is 0. These tiles have enough attachment strength to be stable when both are present, however, since the tiles only have strength 1 bonds, they may not attach alone. This initially prevents the growth of the other bit, which is not placed until the computation is complete, further elaboration of this process is described in section 3.5.

### 3.2 Wires and NOT Gates

Bit information is represented and transferred using a wire. A wire is constructed using two rows of tiles (Figure 2a), each representing a binary value of 0 or 1. This dual rail system initially grows only one of the rows from the input assembly based off the input and then builds into the gates. Before the circuit finishes growing, only one row of each wire will be constructed, and at the end, the other wire row will be built.

Gates such as the NOT grow off the wires. An example of a NOT gate can be seen in Figure 2b, notice how we utilize the third dimension to cross the wires over each other. This gate swaps the position of the rows of tiles; a row that represents a 0 will now be in the upper row and represent a 1. At the end of each gate is a diode gadget that was used in previous work [9]. The gadget is a $2 \times 2$ subassembly that grows only in one direction. If the first tile is placed, the whole thing will be first. If the last tile is placed, nothing else grows since it connects using two strength 1 glues. This prevents errors caused by "backward" growth.


**(a)** Dual Rail Logic. **(b)** NOT Gate.

**Figure 2** (a) We use dual rail gates. The input glue of 1 grows the orange tiles and 0 grows the blue. (b) A NOT gate is implemented by crossing the wires over each other.

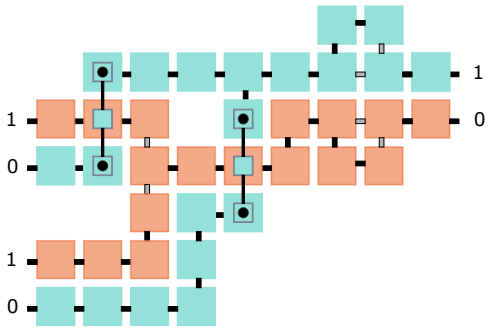



**Figure 3** Full NAND Gate construction in the full circuit. The tiles in orange represent tiles that will be built from an input of 1 input, while the blue tiles come from an input of 0.

## 3.3 NAND Gates

We construct a NAND gate using the NOT gate and cooperative binding. The full NAND gate can be seen in Figure 3. If either input to a NAND gate is 0, the output is always 1. This can be seen in Figures 4a, 4b, and 4c. If any blue tile is placed, the 1 output of the gate will be built. If both inputs are 1, the 0 output can be constructed using cooperative binding.

One thing to note in the case of one output being 0 and the other being 1 is that the blue tiles will be placed along the other wire. However, this will not cause any issues since it can only build back up to the output of the previous gate due to the diode gadget.

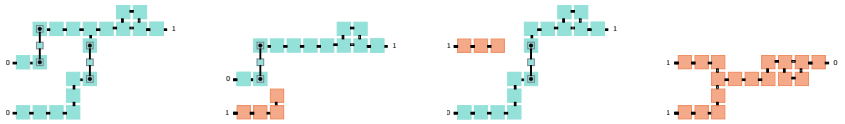
## 3.4 Fan Out and Crossover

Two other gadgets that assist in creating circuits are the fan out and crossover gadgets. The fan out (Figure 5a) splits a wire in order to copy the value to two gates. It does this by having each tile path split, and then use the third dimension to swap the positions.

The crossover gadget (Figure 5c) allows for the creation of non-planar circuits. Using the third dimension, a wire can go over another wire in order to reach its input. While such 3D crossovers simplify constructions greatly, we note that such crossovers are not necessarily needed, as planar circuits can simulate such crossovers using XOR gates [9].

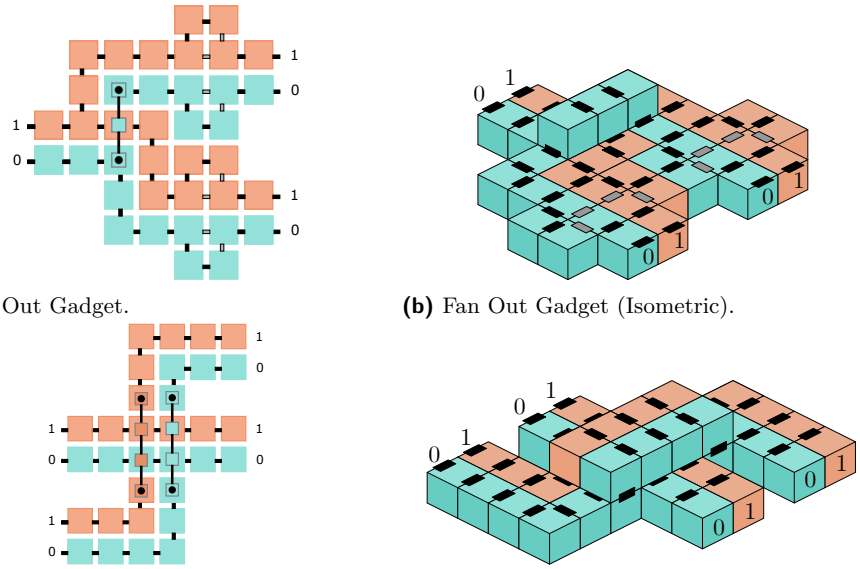
## 3.5 Backfilling and Target Assemblies

In order to perform covert computation, there must exist a unique assembly for each output. The gray tile at the end of the circuit in Figure 6a is one of two flag tiles that denotes the output of the circuit. Once this tile is placed, a row of tiles is built back towards the input



**(a)** Input 00. **(b)** Input 01. **(c)** Input 10. **(d)** Input 11.

**Figure 4** Growth of possible inputs to a NAND gate. The gate will stay like this after computing, before the history is hidden.



**(a)** Fan Out Gadget. **(b)** Fan Out Gadget (Isometric).

**(c)** Crossover Gadget. **(d)** Crossover Gadget (Isometric).

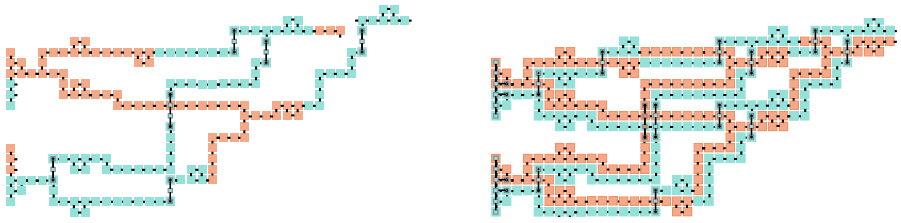
**Figure 5** (a) A fan out gadget. (b) Isometric view of the fan out gadget. (c) While a crossover is not required for universal computation, we can easily implement one by using the 3rd dimension. (d) Isometric view of the crossover gadget.

(Figure 6b). Once the input assembly is reached, the tiles above the input are placed, thus allowing for the input assemblies to be filled in. This causes the entire circuit to be filled out, which hides the original input and computation history.

▶ **Theorem 1.** *For any $n$-bit function $f$ that is computable by a Boolean circuit, there exists a Tile Assembly Computer $\Im$ which covertly computes $f$ in the 3D aTAM with only positive glues. Further, $\Im$ is strictly polynomial in $n$.*

**Proof.** We can construct the tile set $T_c$ from the circuit $c$ that computes $f$. Arrange the gates and wires on the square grid using $\mathcal{O}(n^2)$ space, and scale up each gate and wire by a constant factor. Wires are scaled up by a factor of 2 to account for the dual rail logic wires. The gates are scaled up by a factor depending on which gate it is, however, all the gates we present are only a constant size. This creates assembly $A_{c,Full}$.

We now show that $\Im$ computes $f$. Consider an $n$-bit input $x$ to $f$, using the input template create seed assembly $A_x$. Each gate will grow from $A_x$, computing the circuit on each input. Since backfilling does not occur until the circuit finishes computing, we guarantee only the correct outputs grow from the final gate. The circuit is computed covertly since the output then grows back to the start of the circuit and places the unused inputs. ◀



**(a)** Computed Circuit. **(b)** Output Assembly.

**Figure 6** Example structures of the computation circuit of an XOR using NOTs and NANDs. The circuit before backfilling is on the left, and the final output is shown on the right side. (a) A circuit once the output is computed. (b) Once the output grows backward, the other input bits are placed.

The 3rd dimension is vital in this construction to allow signals to cross over for the NOT gate. Notice the part of the NAND gadget that is computing the AND gate and how the diode uses cooperative binding. Additionally, it would not be possible to build the full input gadget to allow the circuit to backfill. The positions that must be filled will be blocked on one side by the input assembly and on the other by wire. The backfilling here is used differently than in [8] since there each gate would backfill its input wires. There the negative glues were used to allow the tiles to cross over signals to build a NOT gate.

## 4 Exponential Assembly Covert Computer in 2D

In this section, we show that covert computation is possible in 2D in the standard aTAM, where the input can be described in polynomial size, yet the final terminal assembly is exponential in size. Thus, while we are able to achieve strictly polynomial-sized covert computation in 3D, we achieve (non-strict) polynomial-sized covert computation in 2D.

This construction is possible by first computing the function using reversible Toffoli gates, and then replicating and computing the circuit for all possible inputs. Once the output of the original input is placed, the Toffoli gate reverses its computation to build a mirror of the circuit with the input replicated on both the right and left. The output builds an assembly arm used to place tiles on either side of the assembly to increment and decrement the mirrored inputs based on the binary value of the original input, thus seeding a new input for exponential growth in each direction. Thus, for a 4-bit input, it builds the circuit for all $2^4$ possible inputs after it builds the output template.

### 4.1 Toffoli Gate

The Toffoli gate is a 3-bit reversible universal logic gate (Figure 7a), we denote the inputs $A, B, C$, and the outputs $A', B', C'$. The first two input and output bits map to each other: $A = A'$ and $B = B'$. The third output flips the $C$ bit if both $A$ and $B$ are 1. Logically expressed, this is $C \otimes (A \wedge B) = C'$.

We can express an $n$-bit $d$-depth reversible circuit as a $n \times d$ grid where each row represents a wire, and each column is a layer of gates and wires. Each gate can be represented by tiles computing the elementary 2-bit AND and XOR and implementing a fan out, as shown in Figure 7b.

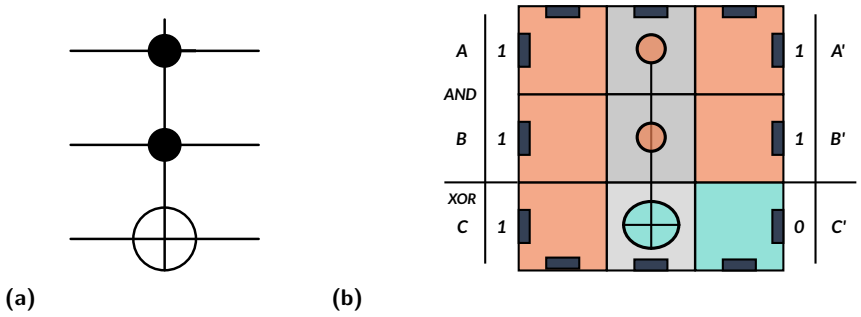



**Figure 7** (a) Logical representation of Toffoli gate. (b) A Toffoli gate on a grid can be represented by the three vertical "cells" of elementary logic gates.

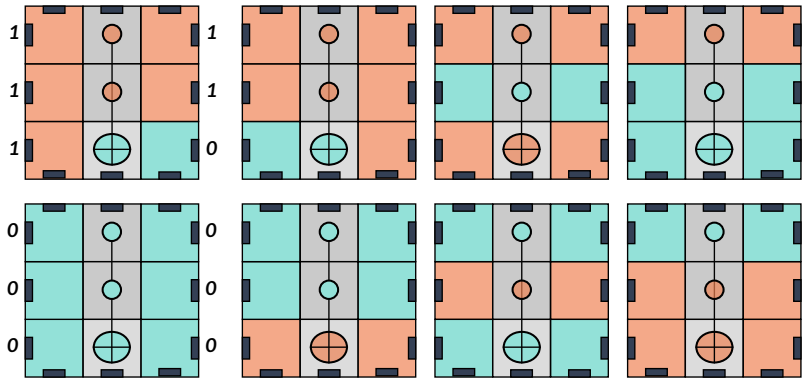


**Figure 8** All possible computations of a single Toffoli gate. 1 (orange), 0 (blue). $111 \to 110, 110 \to 111, 101 \to 101, 100 \to 100$, Row 2: $000 \to 000, 001 \to 001, 010 \to 010$, and $011 \to 011$.

## 4.2 Covert Circuit

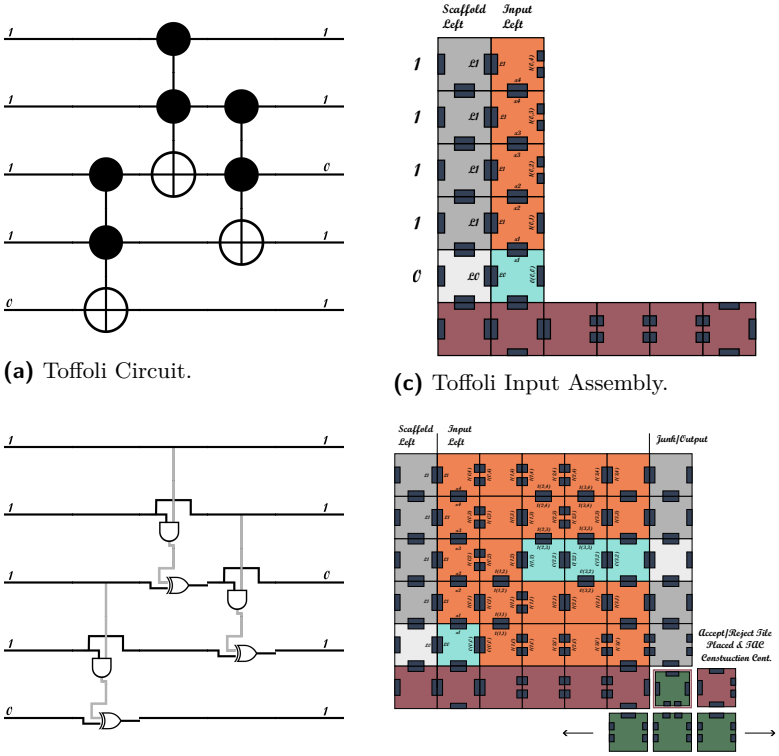
The input template is a specific tile for each bit. Given an $n$-bit string, we create a $n \times 1$ bit assembly with stability-granting left and bottom circuit construction scaffolds, as shown in Figure 9c.

The circuit assembly is a $n \times (d + 2)$ rectangle. Each Toffoli gate is a $3 \times 1$ subassembly. Three possible computations of a single Toffoli gate are shown in Figure 8. Typically, these gates must be reversible, meaning the circuit may grow from the east or west but produce the same assembly We note that the gate itself is not covert, and the "covertness" comes from the full construction.

An example Toffoli circuit is shown in Figure 9a along with the logical representation in Figure 9b. A constructed circuit assembly in one direction can be seen in Figure 9d.

## 4.3 Increment/Decrement Input to Next Circuit Logic

After completion of a circuit, three columns of tiles are built: mark for increment (left), copy or flip (center), and mark for decrement (right). The order of growth of these columns depends on the starting direction. Growing from the left to increment input to the next circuit or from the right to decrement it. Cooperatively with those columns, below the output arm begins its extension to transmit the outcome, accept or reject, of the original circuit. This arm extension continues to the center circuit output outcome tile location. From here, the circuit construction scaffold, previously provided in the input template, may loop back to




**(a)** Toffoli Circuit.

**(c)** Toffoli Input Assembly.

**(b)** AND/XOR Diagram.

**(d)** Toffoli Circuit Assembly.

**Figure 9** (a) Example 5-bit Toffoli Circuit. (b) The Toffoli circuit represented with AND and XOR gates. (c) Example Input Assembly. For each bit (1 or 0), we place the scaffold (grey or white) and input bit tile (orange or blue). The bottom is a row of circuit construction scaffold tiles (maroon). (d) The Toffoli Circuit Assembly built in one direction. The (green) tile below the output/junk column represents the (positive) output and will allow the output control row to place.

the edge of the circuit so the new input scaffold and bits may place as illustrated in Figure 11. The circuit growth continues normally from that point forward, with the exception of the output tile placement.

## 4.4 Output Assembly

Once the output is built, the rows below have $d$ tiles attached in the east and west directions that encode the output. Through cooperative attachment, tiles are placed to allow the strings to increment/decrement, as described above. The final terminal assembly contains every possible computation.

▶ **Theorem 2.** *For all functions $f(x)$ that are computable by a $n$-bit reversible circuit $R$, there exists a polynomial tile assembly computer $\Im = (T, I, O, 2)$ that covertly computes $f(x)$ and has an output assembly of size $\mathcal{O}(2^n)$.*

**Proof.** If there exists a $n$-bit reversible circuit $R$ that computes $f(x)$, we construct tile assembly computer $\Im = (T, I, O, 2)$ as follows. From the circuit $R$ that computes $f$, we design a circuit $R'$ to compute $f$ with Toffoli gates as described in section 4.2. Using $R'$ and the developed input increment/decrement logic for circuit replication, we construct a tile set $T_c$.

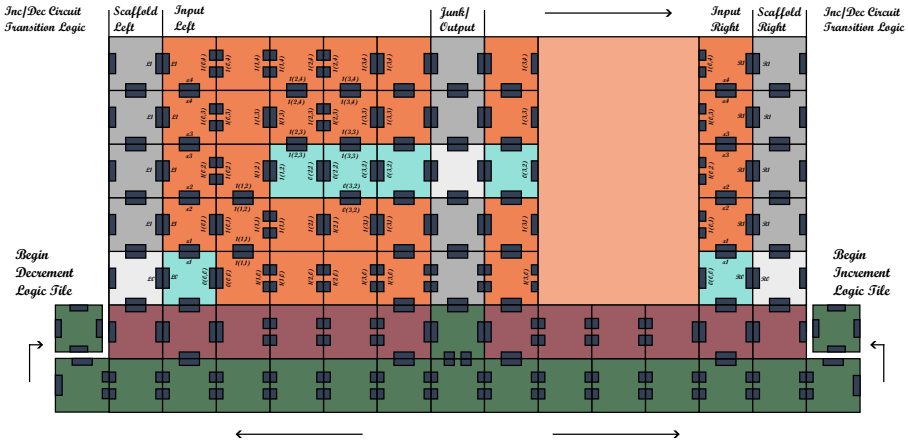



**Figure 10** An example of a symmetrical circuit that has built both sides and is placing begin decrement and increment logic tiles.

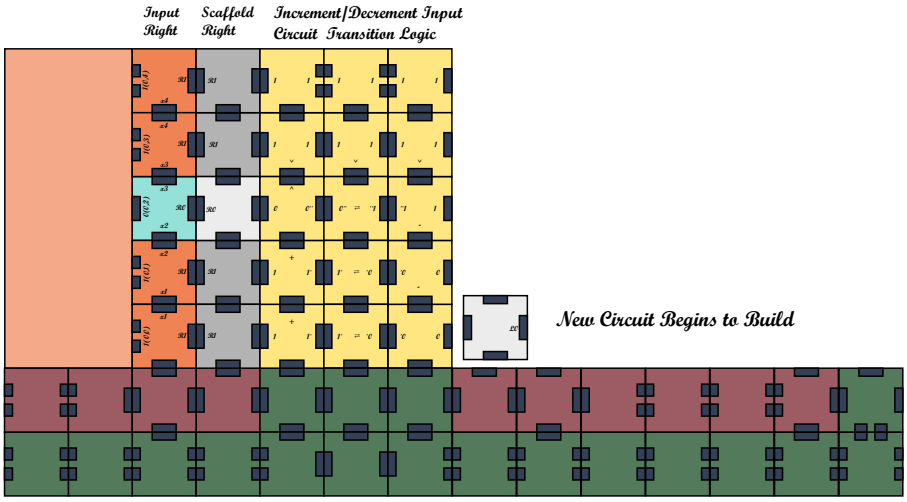



**Figure 11** An example of a new circuit created by incrementing the output from a previously built assembly.

We create the input assembly $I$ by converting the $n$-bit input string $x$ to tiles $L_i$ in scaffold left (figure 9c) and associated input, and a bottom row of tiles called the left circuit construction scaffold.

From here, the left assembly will grow into figure 9d, once the output is determined to be "accept" or "reject", the output indicator tile is placed, and the original output indicator arms grow to allow the Right Assembly the ability to grow as well as place begin decrement and increment logic tiles on the bottom left and right sides of the completed assembly respectively, as seen in figure 10.

All $n$-bit computations of $f(y)$ for $y$ less than original input $x$ will be computed to the left of the original assembly, and all $x_n > x$ after being decremented and incremented using the reversible and symmetric logic in yellow from figure 11. Growth is halted by the INC/DEC logic at overflow in either direction.

The ability to grow further left/right circuit construction scaffolds is dependent on the output arms from the original output indicator arms growing to the center of the circuit about to begin construction where the output accept/reject indicator tile would place, preserving the output status for every circuit built in the TAC.

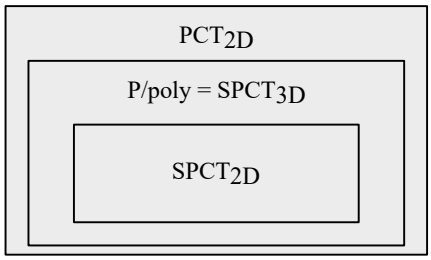



**Figure 12** Diagram showing important classes defined in this section and their relation to P/poly. Note that none of these containments are known to be proper.

As there are only two possible assemblies that can be built, accept all or reject all, the Tile Assembly Computer is polynomial size in description and exponential in output size. ◀

We have shown that if the output assembly is allowed to be exponential in size, that covert computation is possible in the aTAM, even in two dimensions. However, in practice, this is not usually a plausible solution. Given that Unique Assembly Verification is in P [2], it is unlikely that covert computation is possible with a strictly polynomial-size TAC.

▶ **Conjecture 3.** *There does not exist a strictly polynomial-size Tile Assembly Computer in the 2D Abstract Tile-Assembly Model.*

## 5 Polynomial-Sized Covert Circuits

In this section, we define and investigate complexity classes based on decision problems computable by polynomial-sized covert computers. We start by introducing the class P/poly and defining three classes of covertly computable problems: the class of problems covertly computable by a strictly polynomial 3D system ($\text{SPCT}_{3D}$), the class of problems computable by a strictly polynomial 2D system ($\text{SPCT}_{2D}$), and the class of problems computable by a (non-strict) polynomial 2D system ($\text{PCT}_{2D}$). We show how these classes relate to each other, including the result that P/poly is equal to $\text{SPCT}_{3D}$. Our results in this section are summarized in Figure 12.

### 5.1 Complexity Classes

The class P/poly is a well-studied complexity class defined as the class of problems solvable by a polynomial-sized circuit. One note about this class is it puts no requirement on the circuit other than that it exists. This has an equivalent definition as the problems solvable by a polynomial-time Turing machine with a polynomial advice string. We can think of this as the Turing machine being given a description of the circuit and evaluating it. Here, the advice string or circuit must be identical for all inputs of length $n$. [1]

▶ **Definition 4** (P/poly). *The class of problems solvable by a polynomial-sized Boolean circuit. Alternatively, defined as the problems solvable by a polynomial-time Turing machine $M < x, a_{|x|} >$, where $x$ is the input and $a_{|x|}$ is an advice string that is based only on the length of $x$. That is, if two inputs $x, y$ have the same size $|x| = |y|$, then they must use the same advice string.*

[1] Under this definition, every unary language is in this class, including UHALT.

We define the following three complexity classes to categorize the functions that are computable by polynomial-size covert TACs.

▶ **Definition 5** ($\mathsf{SPCT}_{\mathsf{3D}}$). *The class of problems solvable by a strict polynomial sized covert tile assembly computer in the 3D Abstract Tile-Assembly Model.*

*Formally, a language $L$ is in $SPCT_{3D}$ if there exists a sequence of covert TACs $C = \{C_1, C_2, ...\}$ such that the $i^{th}$ TAC, $C_i$, is strictly polynomial in $i$ and if it correctly computes all $x \in L$ where $|x| = i$.*

▶ **Definition 6** ($\mathsf{SPCT}_{\mathsf{2D}}$). *The class of problems solvable by a strict polynomial sized covert tile assembly computer in the 2D Abstract Tile-Assembly Model.*

▶ **Definition 7** ($\mathsf{PCT}_{\mathsf{2D}}$). *The class of problems solvable by a polynomial sized covert tile assembly computer in the 2D Abstract Tile-Assembly Model.*

## 5.2 Strict Polynomial Size Equivalence

To show equivalence between P/poly and $\mathrm{SPCT}_{3\mathrm{D}}$, we first define the 2-Promise Unique Assembly Verification problem, a modified version of Unique Assembly Verification where we are given two assemblies, $a$ and $b$, rather than a single target. The problem asks to separate two cases: accept if an assembly containing $a$ as a subassembly is produced, and reject if an assembly containing $b$ is produced. We assume it is promised that one of these cases is true. This problem is solvable in polynomial time since you only need to attach tiles until one of the two assemblies is produced (Lemma 9).

▶ **Definition 8** (2-Promise Unique Assembly Verification problem). ***Input:*** *Assemblies $a, b$ and an aTAM system $(T, s, \tau)$ which is promised to uniquely produce one of two assemblies, $A$ or $B$, such that $a \subseteq A$ and $b \subset B$.* ***Output:*** *"Yes", if $\Gamma$ uniquely assembles $A$, and "No", if $\Gamma$ uniquely assembles $B$.*

▶ **Lemma 9.** *The 2-Promise Unique Assembly Verification problem is solvable in polynomial time in the 3D aTAM.*

**Proof.** Call greedy grow (from [2]) to get maximal producible assembly $C$. If $\Gamma$ uniquely assembles $C$ and $a \subseteq C$, return "yes". Otherwise, return "no". ◀

Equipped with the algorithm for the 2-promise problem, and taking the description of a covert computer as an advice string, it follows that we can compute the seed assembly from the input template, and the two possible output assemblies from the output template, and then run the algorithm for the 2-Promise UAV problem (Lemma 10). This puts any problem solvable by a polynomial-sized covert circuit in the class P/poly. The other direction of equivalence is given by the 3D covert computer constructions.

▶ **Lemma 10.** *If a language $L$ is computable by a strict polynomial-sized covert tile assembly computer in the 3D aTAM, then $L$ is in P/poly.*

**Proof.** Let $\Im_n(T, I, O, \tau)$ be the covert computer for the strings in language $L$ of size $n$. Since $\Im_n$ is of strict polynomial size, we can encode the tile set, input/output templates, and temperature in $poly(n)$ bits. Thus, $\Im_n$ will be our advice string for membership in P/poly. Further, we are only considering decision problems. Thus, there are only two output templates which we denote as $a_a$ and $b_r$ for accept and reject, respectively.

Consider a Turing machine given the string $x$ and covert circuit $\Im_{|x|} = (T, I, (a_a, b_r), \tau)$ that does the following:

- Convert $x$ to an assembly $I(x)$ using the input template.
- Call the algorithm for 2-Promise UAV on input $((T, I(x), \tau), a_a, b_r)$.
- If the algorithm accepts then $x \in L$, else $x \notin L$

This Turing machine essentially runs the covert computer on $x$ and then checks the output by seeing which template is included in the final assembly. ◀

▶ **Theorem 11.** *The classes $SPCT_{3D}$ and P/poly are equivalent.*

**Proof.** By Lemma 10, if a language is in P/poly there is a Boolean circuit of polynomial size which computes it, giving us P/poly $\subseteq$ PCT$_{3D}$. In Theorem 1 we show that if there exists a Boolean circuit, there exists a strictly polynomial sized covert computer that computes the circuit. ◀

## 5.3 Polynomial Sized 2D Covert Circuits

Here, we use previous constructions to show that the class of polynomial sized 2D covert circuits is at least as strong as strict polynomial covert circuits. That is every language in $SPCT_{3D}$ is in $PCT_{2D}$.

▶ **Theorem 12.** *If a language $L$ is in P/poly then $L$ is in $PCT_{2D}$*

**Proof.** In Lemma 10 we show that if a language is in P/poly there is a Boolean circuit of polynomial size which computes it. Any Boolean circuit can be turned into a reversible circuit, thus by Theorem 2, if there exists a reversible circuit, there exists a polynomial tile assembly computer that computes it in 2D. ◀

# 6 Conclusion and Future Work

Previous work in the aTAM required negative glues in order to build covert Tile Assembly Computers. We have provided two new covert computers in the aTAM with only positive glue strengths, one in (just-barely) 3D and one in 2D with an exponential-sized output assembly. These covert TACs add new tools to the field that may find use in future complexity results, or in future applications related to privacy, cryptography, or biological computation. We have further initiated the study of covert computers in the context of known complexity classes, showing connections to the well-studied class P/poly. These results motivate future work to find functions that can be covertly computed in the 2D aTAM with strict polynomial size, such as (perhaps) Branching Programs.

Some additional specific directions for future work are as follows. We show the containment of the class of strict polynomial computers to be in P/poly. Can this be improved? Could we possibly use the P/poly log space analogue L/poly? What about in smaller classes, such as covert computers with non-cooperative binding or at temperature-1?

## References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Scott D. Kominers, and Robert Schweller. Shape replication through self-assembly and rnase enzymes. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'10, pages 1045–1064, 2010. doi:10.1137/1.9781611973075.85.

2 Leonard M. Adleman, Qi Cheng, Ashish Goel, Ming-Deh A. Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.

3 Andrew Alseth and Matthew J. Patitz. The need for seed (in the abstract tile assembly model). In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA'23, pages 4540–4589, 2023.

4 Spring Berman, Sándor P Fekete, Matthew J Patitz, and Christian Scheideler. Algorithmic foundations of programmable matter (dagstuhl seminar 18331). In *Dagstuhl Reports*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

5 Yuriy Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Comp. Sci.*, 378:17–31, 2007.

6 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Covert computation in staged self-assembly: Verification is pspace-complete. In *Proceedings of the 29th European Symposium on Algorithms*, ESA'21, 2021.

7 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Verification and computation in restricted tile automata. *Natural Computing*, 2021. doi:10.1007/s11047-021-09875-x.

8 Angel A. Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert Computation in Self-Assembled Circuits. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:14, 2019.

9 Angel A. Cantu, Austin Luchsinger, Robert T. Schweller, and Tim Wylie. Covert computation in self-assembled circuits. *Algorithmica*, 83:531–552, 2019.

10 Luis Ceze, Jeff Nivala, and Karin Strauss. Molecular digital data storage using dna. *Nature Reviews Genetics*, 20(8):456–466, 2019.

11 Cameron Chalk, Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of any shape with constant tile types using high temperature. In *Proc. of the 26th Annual European Symposium on Algorithms*, ESA'18, 2018.

12 Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018. doi:10.1007/s00453-017-0318-0.

13 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

14 Supreme Court. Ass'n for molecular pathology v. myriad, 2013.

15 Erik Demaine, Matthew Patitz, Robert Schweller, and Scott Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor. *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, 9, January 2010. doi:10.4230/LIPIcs.STACS.2011.201.

16 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017. Computational Self-Assembly. doi:10.1016/j.tcs.2016.11.020.

17 David Doty, Lila Kari, and Benoît Masson. Negative interactions in irreversible self-assembly. *Algorithmica*, 66(1):153–172, 2013.

18 David Doty, Jack H Lutz, Matthew J Patitz, Robert T Schweller, Scott M Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 302–310. IEEE, 2012.

19 Pim WJM Frederix, Ilias Patmanidis, and Siewert J Marrink. Molecular simulations of self-assembling bio-inspired supramolecular systems and their connection to experiments. *Chemical Society Reviews*, 47(10):3470–3489, 2018.

20 David Furcy, Samuel Micka, and Scott M. Summers. Optimal program-size complexity for self-assembled squares at temperature 1 in 3d. *Algorithmica*, 77(4):1240–1282, March 2016. doi:10.1007/s00453-016-0147-6.

**21** David Furcy, Scott M. Summers, and Logan Withers. Improved Lower and Upper Bounds on the Tile Complexity of Uniquely Self-Assembling a Thin Rectangle Non-Cooperatively in 3D. In Matthew R. Lakin and Petr Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.27.4.

**22** Daniel Hader, Aaron Koch, Matthew J Patitz, and Michael Sharp. The impacts of dimensionality, diffusion, and directedness on intrinsic universality in the abstract tile assembly model. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2607–2624. SIAM, 2020.

**23** Adam M Kabza, Nandini Kundu, Wenrui Zhong, and Jonathan T Sczepanski. Integration of chemically modified nucleotides with dna strand displacement reactions for applications in living systems. *Wiley Interdisciplinary Reviews: Nanomedicine and Nanobiotechnology*, 14(2):e1743, 2022.

**24** Alexandra Keenan, Robert Schweller, Michael Sherman, and Xingsi Zhong. Fast arithmetic in algorithmic self-assembly. *Natural Computing*, 15(1):115–128, March 2016.

**25** Paul WK Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468, 2000.

**26** Robert Schweller, Andrew Winslow, and Tim Wylie. Complexities for high-temperature two-handed tile self-assembly. In Robert Brijder and Lulu Qian, editors, *DNA Computing and Molecular Programming*, pages 98–109, Cham, 2017. Springer International Publishing.

**27** Heribert Vollmer. *Introduction to Circuit Complexity*. Springer Berlin Heidelberg, 1999. doi:10.1007/978-3-662-03927-4.

**28** Klaus F Wagenbauer, Christian Sigl, and Hendrik Dietz. Gigadalton-scale shape-programmable dna assemblies. *Nature*, 552(7683):78–83, 2017.

**29** Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

# Restless Exploration of Periodic Temporal Graphs

**Thomas Bellitto**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Cyril Conchon–Kerjan**
DIENS, Ecole normale supérieure, F-75005 Paris, France

**Bruno Escoffier**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
Institut Universitaire de France, Paris, France

## Abstract

A temporal graph is a sequence of graphs, indexed by discrete time steps, with a fixed vertex set but with an edge set that is able to change over time. In the temporal graph exploration problem, an agent wants to visit all the vertices of a given temporal graph. In the classical model, at each time step the agent can either stay where they are, or move along one edge. In this work we add a constraint called *restlessness* that forces the agent to move along one edge at each time step. We mainly focus on (infinite) periodical temporal graphs. We show that if the period is 2 one can decide in polynomial time whether exploring the whole graph is possible or not, while this problem turns out to be NP-hard for any period $p \geq 3$. We also show some time bounds on the explorations of such graphs when the exploration is possible.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** Temporal graphs, Graph exploration, NP-completeness

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.13

## 1 Introduction

A temporal graph is a sequence of graphs $G = (G_1, G_2, \dots, G_k, \dots)$, where $G_t = (V, E_t)$ is called the snapshot at time step $t$. Vertices remain but edges are susceptible to be removed or added at each time step. When the sequence is finite, the number of steps is called the *lifetime* of the temporal graph. The study of algorithmic aspects of temporal graphs was promoted lately due to the emergence of dynamic networks with change of links over time (e.g., social-, wireless mobile-, transportation networks). Among the several problems that have been studied, the *temporal exploration problem* (TEXP) has received a lot of attention in the last decade. In this problem, an agent aims at visiting all vertices of $V$, in minimal time if visiting all vertices is possible. In the classical model, called the strict variant, the agent can travel on at most one edge at each time step, while in the non-strict variant the agent can use as many edges as they want at each time step [7].

In the strict model, while determining whether it is possible or not to explore the graph (i.e., visit all vertices) is NP-hard [10], it is not hard to see that this is always possible when (1) the graph is connected at each time step and (2) the lifetime is at least $n^2$, where $n = |V|$. A substantial amount of works have been devoted to study the problem on some specific graph classes, both on the computational complexity and on possible improvements of this $O(n^2)$ bound on the lifetime of the graph (which is in fact tight - lifetime $\Omega(n^2)$ might be necessary - for general graphs), see for instance [3, 5, 6, 7, 9]. As a notable example, if each snapshot is connected and of bounded degree, then $O(n^{1.75})$ steps are sufficient to explore the graph [6]. Interestingly, the same upper bound holds in general (connected) graphs in a slightly different model, where the agent is allowed to make at most two moves per step, instead of at most one.


© Thomas Bellitto, Cyril Conchon–Kerjan, and Bruno Escoffier;
licensed under Creative Commons License CC-BY 4.0

However, the results obtained in previous works do not apply anymore under the constraint called *restlessness*. A $\Delta$-restless temporal path is a path where the agent walking in the graph is not allowed to wait for $\Delta$ steps of time at a vertex before making their next move. For example, this can be used to model non delay tolerant network where a packet can only be stored for a limited amount of time in the nodes, because of memory limitation [4]. Another recent application is the study of paths in virus infection, where a virus can only spread if it infects somebody new before the infected person recovers [8].

We also would like to point out that restless walks in temporal graphs also offer a powerful generalization of properly-colored walks in edge-colored graphs. If the edges of a graph are colored, a walk is said to be properly-colored iff it does not use two edges of the same color consecutively. These graphs themselves offer an interesting generalization of directed graphs (even undirected edge-colored graphs) and there is a rich literature around problems of properly-colored spanning paths or trails (see Chapter 11 of [1]). Given a $k$-edge colored graph, one can build a periodic temporal graph, with infinite lifetime, where the edges available at timeframe $i \mod k$ are exactly the edges colored $i$. If walks have to be $k$-restless, an agent at a vertex $v$ can use edges of any color for their next move, except the one they used to get to $v$, which will take too long to appear again.

In this article, we consider the 1-restless variant of TEXP, i.e. the variant where waiting at a vertex is not possible and the agent therefore has to make exactly one move per step. We denote it by 1-RTEXP. As it is the most restless case, we believe that it is a strong starting point to study the impact of restlessness on the explorability of a graph.

It is easy to see that exploring a temporal graph restlessly can be much more difficult than when we may wait. As a matter of fact, this may not be possible even in a temporal graph with connected snapshots and infinite lifetime, as shown in the simple Example 1.

▶ **Example 1.** For $i > 0$, $G_i$ is the graph on the left of Figure 1 if $i$ is odd, it is the graph on the right of Figure 1 if $i$ is even. Hence, the graph has infinite lifetime, and it is 2-periodical. If the starting vertex (at $t = 1$) is one vertex among $\{D, E, F\}$, then the agent, being forced to move, will be in $\{A, B, C\}$ at time 2, back in $\{D, E, F\}$ at time 3, and so on. It will never be able to visit node $H$.

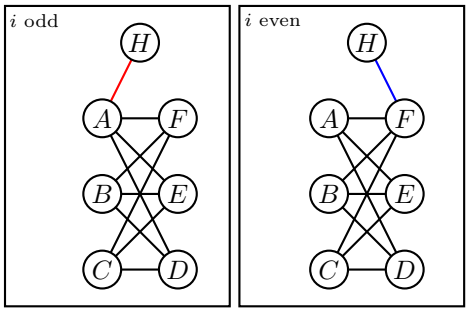



**Figure 1** Snapshots at odd and even timesteps. We use colors to highlight the difference.

Note that the same example works with replacing the subgraph induced by $\{A, B, C, D, E, F\}$ with any (connected) bipartite graph. Following this, a natural question is to find either sufficient conditions for a temporal graph to be explorable restlessly, or tractable cases where 1-RTEXP can be determined in polynomial time. As exploration is not guaranteed even if the lifetime is infinite (and the snapshots are connected), we focus in this work on periodic graphs of infinite lifetime, which are one of the most natural classes of graph of

infinite lifetime that we can encode in finite space (an essential condition for complexity to even make sense). Our main result is to provide a sharp separation between tractability and NP-hardness of 1-RTEXP, based on the period of the graph, summarized in the two following theorems.

▶ **Theorem 2.** *In 2-periodical temporal graphs, 1-RTEXP is polynomially solvable.*

▶ **Theorem 3.** *For any $p \geq 3$, 1-RTEXP is NP-hard in $p$-periodical temporal graphs, even if each snapshot is connected.*

We complement these results by showing some sharp bounds in the number of iterations needed to explore such graphs (whenever exploration is possible).

The article is organized as follows. We give a formal definition of the problem in Section 2. Theorems 2 and 3 are shown respectively in Sections 3 and 4. Exploration time bounds are given in Section 5.

## 2 Preliminaries

▶ **Definition 4.** *A temporal graph is a sequence $G = (G_1, G_2, \dots)$ of graphs $G_t = (V, E_t)$. If the sequence is finite, the length of the sequence is called the lifetime of the graph (otherwise the lifetime is infinite). $G_t$ is called the snapshot of $G$ at time $t$. A (infinite) temporal graph is $p$-periodical if $G_{i+p} = G_i$ for every $i \geq 1$.*

Note that a $p$-periodical temporal graph is fully described by giving the first $p$ snapshots $G_1, \dots, G_p$, thus with a description of (finite) size $O(pn^2)$.

▶ **Definition 5.** *In a temporal graph, a* temporal walk *or* journey *is a sequence of vertices $(v_i, v_{i+1}, \dots, v_j)$ such that for all $k$, $v_k = v_{k+1}$ or $(v_k, v_{k+1}) \in E_k$. The vertices $v_i$ and $v_j$ are called respectively the start vertex and the end vertex and $i$ and $j$ respectively the starting time and ending time. A 1-restless journey, which we simply call a restless journey in this paper, is a journey where $v_k \neq v_{k+1}$, for all $k = i, \dots, j-1$.*

▶ **Definition 6** (1-RTEXP)**.** *Given a temporal graph $G$ and a start vertex $s \in V$, 1-RTEXP asks whether there is a restless journey starting at $s$ at time 1 that contains all vertices of the graph.*

If so, we say that the (temporal) graph is fully explorable, when starting at $s$.

Dealing with periodical temporal graphs, the case of period 1 is trivial, as when $p = 1$ we have $G_{i+1} = G_i$, so the graph is somehow static. Hence, it is fully explorable if and only if it is connected.

Before starting our results, let us mention a result on reachability in $p$-periodical temporal graphs, which easily follows from classical BFS in graphs.

▶ **Lemma 7** (Reachability)**.** *Given a $p$-periodical temporal graph, two vertices $u$ and $w$, and two indices $i, j \in \{1, \dots, p\}$, we can determine in linear time if there exists a restless journey starting at $u$ at some time $t \equiv i \bmod p$ and ending at $w$ at some time $t' \equiv j \bmod p$. Moreover, if such a journey exists, there exists one with length (number of edges) at most $np - 1$.*

**Proof.** We build a graph $G'$ on $np$ vertices $(v, k)$ for $v \in V$ and $k \in \{1, \dots, p\}$. In $G'$ we put an arc from $(v, k)$ to $(v', k+1)$ (or to $(v', 1)$ if $k = p$) if there is an edge $(v, v')$ in snapshot $G_k$. Then, a restless journey in $G$, starting at $u$ at some time $t \equiv i \ mod \ p$ and ending at $w$ at some time $t' \equiv j \ mod \ p$, corresponds to a walk in $G'$ starting at $(u, i)$ and ending at $(w, j)$. The existence of such a walk can be determined using a BFS on $G'$.

As $G'$ has linear size (with respect to the input), this can be checked in linear time. Moreover, if such a path exists, there exists a simple one, thus with length at most $np-1$. ◀

If this is the case, we will say that $(v, j)$ *is accessible, or reachable, from* $(u, i)$.

## 3 2-periodical temporal graphs

This section is dedicated to the proof of Theorem 2 which states that 1-RTEXP is polynomially solvable in 2-periodical temporal graphs.

To prove this, we reduce the problem to 2-Sat (restriction of Sat on clauses of size at most 2), which is well known to be polynomial time solvable. The rough idea is to consider two variables per vertex, one saying that we will visit the vertex at an odd time step, the other one saying that we will visit the vertex at an even time step (so at least one of them should be true). We also introduce some variables that represent the order in which we will visit the vertices - more precisely the order of the *first* visit of vertex $v$ at odd and/or even time steps. Corresponding constraints are built thanks to the reachability lemma (Lemma 7). Additional constraints ensure the global feasibility and the fact that the journey starts at $s$.

Let us now formally define the 2-Sat formula. We are given a 2-periodical temporal graph $G$ (i.e., its 2 snapshots $G_1$ and $G_2$), and one start vertex $s \in V$. We construct the following 2-Sat formula $F(G, s)$ :

- Variables:
  - For each vertex $u$ in $V$ we create two variables $u_1$ and $u_2$. As explained above, the variable $u_i$, $i \in \{1, 2\}$, will be true if we visit $u$ at time parity $i$ in our exploration. Let $I$ be the set of variables.
  - For each pair $u_i, v_j$ (for $u, v \in V$, $i, j \in \{1, 2\}$, possibly $u = v$ and/or $i = j$) we create a variable $u_i \rightsquigarrow v_j$. In our construction this variable is true if (1) we visit $u$ at time parity $i$, and (2) either we do not visit $v$ at time parity $j$, or the first visit of $u$ at time parity $i$ is before the first visit of $v$ at time parity $j$.
- Clauses:
  - (VISIT) For each vertex $v$ in $G$ we construct a clause $(v_1 \vee v_2)$, meaning that we have to visit $v$ at time parity 1 or 2 in order to visit the whole graph.
  - (REACH) For each pair $(u_i, v_j)$, if there is *no* restless journey from $u$ at time parity $i$ to $v$ at time parity $j$, we create a clause $((v_j \rightsquigarrow u_i) \vee (\overline{v_j}))$ meaning that we either visit $v$ at time parity $j$ before $u$ at time parity $i$ or do not visit $v$ at time parity $j$ at all.
  - (ORDER) For each pair $u_i, v_j$ in $I$ we create the clause $(\overline{u_i \rightsquigarrow v_j} \vee \overline{v_j \rightsquigarrow u_i})$ ensuring that we do not claim to visit $v$ at time parity $j$ before we visit $u$ at time parity $i$ and at the same time to visit $u$ at time parity $i$ before $v$ at time parity $j$.
  - (START) We create a clause $(s_1)$ for the start vertex $s$ at time parity 1 meaning that we have to go through this state.
  - (FIRST) For other $u_j$ in $I$ we create the clause $(s_1 \rightsquigarrow u_j)$ meaning that we visit $s$ at time 1 before any other vertex, *i.e.* that we start our exploration on $s$ at time odd.

Note that the formula has $2n + 4n^2$ variables and $O(n^2)$ clauses.

We now show in Lemmas 8 and 11 that the temporal graph is fully explorable from $s$ (at time 1) if and only if $F(G, s)$ is satisfiable. This shows Theorem 2. We note that when the graph is fully explorable from $s$, a corresponding restless journey can easily be built from a truth assignment satisfying $F(G, s)$.

▶ **Lemma 8.** *If the graph $G$ can be fully explored starting from $s$ at time 1 then $F(G, s)$ is satisfiable.*

**Proof.** We set the value of each variable according to a restless journey exploring the whole graph starting from $s$ at time 1, as explained in the description of the variables in the formula.

- Clauses (VISIT) are satisfied as the journey visits all vertices, either at time odd or even.
- If we cannot visit $v$ at time parity $j$ after $u$ at time parity $i$ in $G$, either we do not visit $v$ at time parity $j$ ($\overline{v_j}$ is true), or we visit $v$ at time parity $j$ but not $u$ at time parity $i$ ($\overline{v_j \rightsquigarrow u_i}$ is true), or we visit both and then necessarily $v$ at time parity $j$ before $u$ at time parity $i$ ($\overline{v_j \rightsquigarrow u_i}$ is true). Then, clauses (REACH) are satisfied.
- Clauses (ORDER) are verified since following the journey gives a strict order of first visits (note that if $u$ is not visited at time parity $i$ then $u_i \rightsquigarrow v_j$ is false).
- Clauses (START) and (VISIT) are verified since we start our journey from $s$ at time 1. ◀

Assume now that the formula $F(G, s)$ is satisfiable and consider a satisfying truth assignment $S$ of it. Let us define a graph $G_0$ as follows:

- Vertices: For each variable $u_i$ set to true in $S$ the graph contains vertex $u_i$
- Edges: For any two vertices $u_i, v_j$ in the graph, we put the arc $(u_i, v_j)$ if the variable $u_i \rightsquigarrow v_j$ is true in $S$.

▶ **Lemma 9.** *If there is an arc $(u_i, v_j)$ in $G_0$ then $(v, j)$ is accessible from $(u, i)$.*

**Proof.** Assume by contradiction that it is impossible. Then there is by definition in $F(G, s)$ a clause $((v_j \rightsquigarrow u_i) \vee (\overline{v_j}))$. From $v_j \in G_0$ we deduce $v_j$ is true. Thus, to verify the clause, $(v_j \rightsquigarrow u_i)$ must be true. Since we have the above mentioned edge in $G_0$, we also have that $(u_i \rightsquigarrow v_j)$ is true. Consequently, the clause $(\overline{u_i \rightsquigarrow v_j} \vee \overline{v_j \rightsquigarrow u_i})$ of $F(G, s)$ is not satisfied, a contradiction. ◀

Note that by an easy recurrence Lemma 9 shows that when there is a path from $u_i$ to $v_j$ in $G_0$ then $(v, j)$ is accessible from $(u, i)$ in the temporal graph.

▶ **Lemma 10.** *If there is no arc between two vertices $u_i$ and $v_j$ of $G_0$ (neither $(u_i, v_j)$ nor $(v_j, u_i)$), then $(v, j)$ is accessible from $(u, i)$ and vice-versa.*

**Proof.** Assume by contradiction and without loss of generality that $(v, j)$ is not accessible from $(u, i)$. Then there is by definition a clause $((v_j \rightsquigarrow u_i) \vee \overline{v_j})$ in $F(G, s)$. Since $v_j \in G_0$, $v_j$ is true, thus to verify the clause we have that $(v_j \rightsquigarrow u_i)$ is true. By construction, we do have an arc from $u_i$ to $v_j$, a contradiction. ◀

▶ **Lemma 11.** *If $F(G, s)$ is satisfiable then the temporal graph $G$ can be fully explored starting from vertex $s$ at time* 1.

**Proof.** Consider some satisfying assignment $S$ of $F(G, s)$, and the associated graph $G_0$.

Take a topological order of the strongly connected components (SCC) of $G_0$.

This gives us a suitable exploration order of the graph. Indeed, Lemma 9 ensures that if $(u_i, v_j)$ are in the same SCC, then $(v, j)$ is accessible from $(u, i)$ in $G$. If $u_i$ and $v_j$ are in two consecutive SCC in the topological order, then again $(v, j)$ is accessible from $(u, i)$ in $G$: indeed, this follows from Lemma 9 when there is an arc from the SCC of $u_i$ to the one of $v_j$, and from Lemma 10 when there is no arc between the two SCC.

Thus, if we consider an order of vertices of $G_0$ that follow the topological order of SCC, then we can build a journey in the temporal graph that visit all the corresponding vertices. As $G_0$ contains either $u_1$ or $u_2$ for any vertex $u$ of the temporal graph, the journey does explore all the vertices of the temporal graph.

To finish the proof, we shall argue that we can choose a journey that starts at $(s, 1)$. First note that as $s_1$ is true (thanks to the clause (START)) there is a vertex $s_1$ in $G_0$. Moreover, from the clauses (FIRST) we know that there is an arc $(s, u_i)$ for any vertex $u_i$ in $G_0$. Hence, $s_1$ is necessarily in the first SCC in the topological order, and the journey can be chosen to start at $s$ at time 1. ◀

# 4 $p$-periodical temporal graph with $p \geq 3$

We show in this section Theorem 3, i.e., that 1-RTEXP is NP-hard for $p$-periodical graphs, even with connected snapshots, for any $p \geq 3$. We first deal with the case $p = 3$, and then show how the proof can be adapted to the cases $p \geq 4$.

## 4.1 Case $p = 3$

Given a 3-periodical temporal graph $G$ and a start vertex $s$, we study whether the agent can visit all the vertices $V$.

In order to show that the problem is NP-hard, we build a reduction from 3-Sat-(2,2), a restriction of 3-Sat where each variable appears exactly 2 times positively and 2 times negatively. This problem is known to be NP-complete [2].

Given a 3-Sat-(2,2) formula, we build a graph that contains a clause-gadget (described in Section 4.1.1) for every clause, a variable-gadget for every variable (described in Section 4.1.2) and a trap-gadget (described in Section 4.1.3) which makes the snapshots connected, while forcing the agent to explore the graph in some specific order.

### 4.1.1 The clause-gadget

For every clause $C_i$ with literals $\ell_i^1, \ell_i^2, \ell_i^3$, we construct the gadget depicted in Figure 2. We create 8 vertices, $\ell_i^1, \ell_i^2, \ell_i^3, v_i^1, v_i^2, v_i^3, v_i^4, v_i^5$, and 12 edges as follows:

- 3 edges $(v_i^1, \ell_i^j)$, $j \in \{1, 2, 3\}$, present on times $k \equiv 1 \bmod 3$;
- 3 edges $(\ell_i^1, \ell_i^2)$, $(\ell_i^2, \ell_i^3)$, $(\ell_i^3, \ell_i^1)$ present on times $k \equiv 2 \bmod 3$;
- 3 edges $(\ell_i^j, v_i^2)$, $j \in \{1, 2, 3\}$, present on times $k \equiv 3 \bmod 3$;
- 1 edge $(v_i^2, v_i^3)$ present on times $k \equiv 1 \bmod 3$;
- 1 edge $(v_i^3, v_i^4)$ present on times $k \equiv 2 \bmod 3$;
- 1 edge $(v_i^4, v_i^5)$ present on times $k \equiv 3 \bmod 3$.

This construction is illustrated in Figure 2.

Suppose that an agent is at $v_i^1$ at time $k \equiv 1 \bmod 3$ and then moves inside this gadget. The agent will be at one vertex from $\{\ell_i^1, \ell_i^2, \ell_i^3\}$ at time $k + 1$, then on a second vertex from $\{\ell_i^1, \ell_i^2, \ell_i^3\}$ at time $k + 2$ and then joining $v_i^2$ at time $k + 3$. The agent will eventually go through $v_i^3, v_i^4$ and $v_i^5$ in this order as it is the only possible journey. To sum up, the agent will have visited every vertex of the gadget besides one from $\{\ell_i^1, \ell_i^2, \ell_i^3\}$, and will be in $v_i^5$ at $k' \equiv 1 \bmod 3$.

▶ Remark 12. The vertices $\ell_i^1, \ell_i^2, \ell_i^3$ corresponding to literals will be linked to the variable-gadgets. The unvisited vertex will correspond to the literal that will be set to true in the clause in order to verify the formula.

The graph contains one gadget for each clause. More precisely, the gadgets are chained together, where the vertex $v_i^5$ from the gadget of clause $C_i$ is merged with the first vertex $v_{i+1}^1$ of clause $C_{i+1}$. This way, if an agent starts at $v_1^1$ at time $k \equiv 1 \bmod 3$ and moves inside the clause-gadgets, it can traverse all the gadgets and will arrive at $v_m^5$ (where $m$ is the number of clauses) at time $k' \equiv 1 \bmod 3$ and will have visited every vertex except exactly one among $\{\ell_i^1, \ell_i^2, \ell_i^3\}$ for each $i = 1, \ldots, m$.

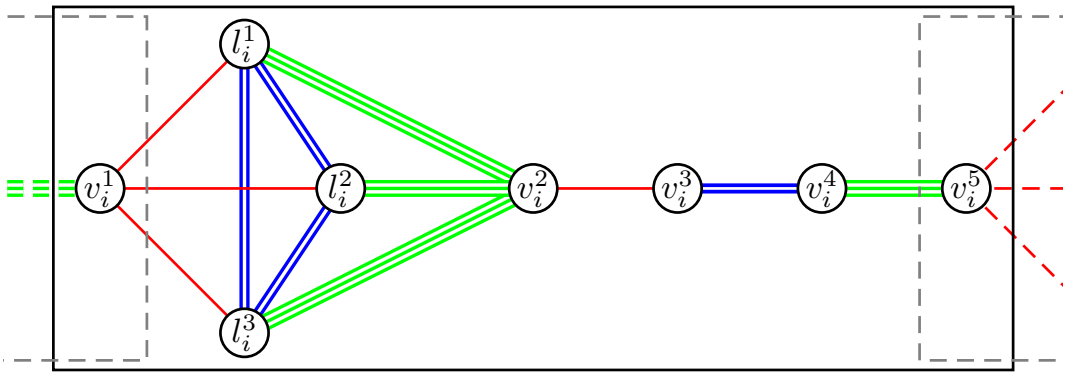



**Figure 2** The gadget associated to the clause $i$. Red edges are present on times $k \equiv 1 \bmod 3$, blue edges on times $k \equiv 2 \bmod 3$ and green edges on times $k \equiv 3 \bmod 3$. For black-and-white readability, the edges present on times 1, 2 and 3 mod 3 are also respectively denoted by simple, double and triple edges.

### 4.1.2 The variable gadget

Every variable $y_i$ appears in 4 clauses, 2 times positively and 2 times negatively. Hence, there are in the clause-gadgets 2 vertices corresponding to $y_i$, and 2 vertices corresponding to $\overline{y_i}$. Then, for every variable $y_i$ we build the gadget depicted in Figure 3: it contains 10 (new) vertices $w_i^j, j = 1, \ldots, 10$, and is linked to the clause-gadgets through the 4 vertices corresponding to $y_i$ and $\overline{y_i}$ (called $y_i^1$, $y_i^2$, $-y_i^1$ and $-y_i^2$ in the figure). Besides the 10 vertices, the gadget contains the following 18 edges:

- Edges $(w_i^3, y_i^1)$, $(w_i^5, -y_i^1)$, $(w_i^4, w_i^8)$, $(w_i^6, w_i^8)$, $(w_i^{10}, y_i^2)$, $(w_i^{10}, -y_i^2)$ and $(w_i^7, w_i^{10})$ present on times $k \equiv 1 \bmod 3$;
- Edges $(w_i^1, w_i^2)$, $(w_i^3, w_i^4)$, $(w_i^5, w_i^6)$ and $(w_i^9, w_i^{10})$ present on times $k \equiv 2 \bmod 3$;
- Edges $(w_i^2, y_i^1)$, $(w_i^2, -y_i^1)$, $(w_i^4, y_i^2)$, $(w_i^6, -y_i^2)$, $(w_i^3, w_i^7)$, $(w_i^5, w_i^7)$ and $(w_i^8, w_i^9)$ present on times $k \equiv 3 \bmod 3$;

This construction is illustrated in Figure 3.

Suppose that an agent is in $w_1$ at some time $k \equiv 2 \bmod 3$ and move inside the variable-gadget (on vertices $w_i^j$, $y_i^1$, $y_i^2$, $-y_i^1$, $-y_i^2$). Then they must go to $w_i^2$, and then can go either to $y_i^1$ or to $-y_i^1$. If they go to $y_i^1$, then they have to go to $w_i^3$ as we consider here the case where they stay inside the gadget (we will show later that they cannot respect the restlessness condition if they leave the gadget). At this point they must go to $w_i^4$, then $y_i^2$ and $w_i^{10}$. They may then visit $w_i^9$, $w_i^8$, $w_i^6$, $w_i^5$, $w_i^7$ and then go back to $w_i^{10}$. Then, either they do the same (now useless) cycle on these vertices, or leave the gadget.

In other words, they can either take $P_1 = (w_i^1, w_i^2, y_i^1, w_i^3, w_i^4, y_i^2, w_i^{10}, w_i^9, w_i^8, w_i^6, w_i^5, w_i^7, w_i^{10})$ (thus visiting the 10 vertices $w_i^j$, and $y_i^1$ and $y_i^2$), or the symmetrical path $P_2 = (w_i^1, w_i^2, -y_i^1, w_i^5, w_i^6, -y_i^2, w_i^{10}, w_i^9, w_i^8, w_i^4, w_i^3, w_i^7, w_i^{10})$ .

Note that in both cases, they are in a vertex in $\{y_i^1, y_i^2, -y_i^1, -y_i^2\}$ only at some time $k' \equiv 1 \bmod 3$.

The variable-gadgets are chained, by merging the vertex $w_i^{10}$ of the gadget of variable $y_i$ to the vertex $w_{i+1}^1$ of the gadget of the variable $y_{i+1}$.

Also, the last vertex $v_m^5$ of the clause-gadgets is linked to the vertex $w_1^1$ of the first variable-gadget by an edge present at time 1 mod 3.

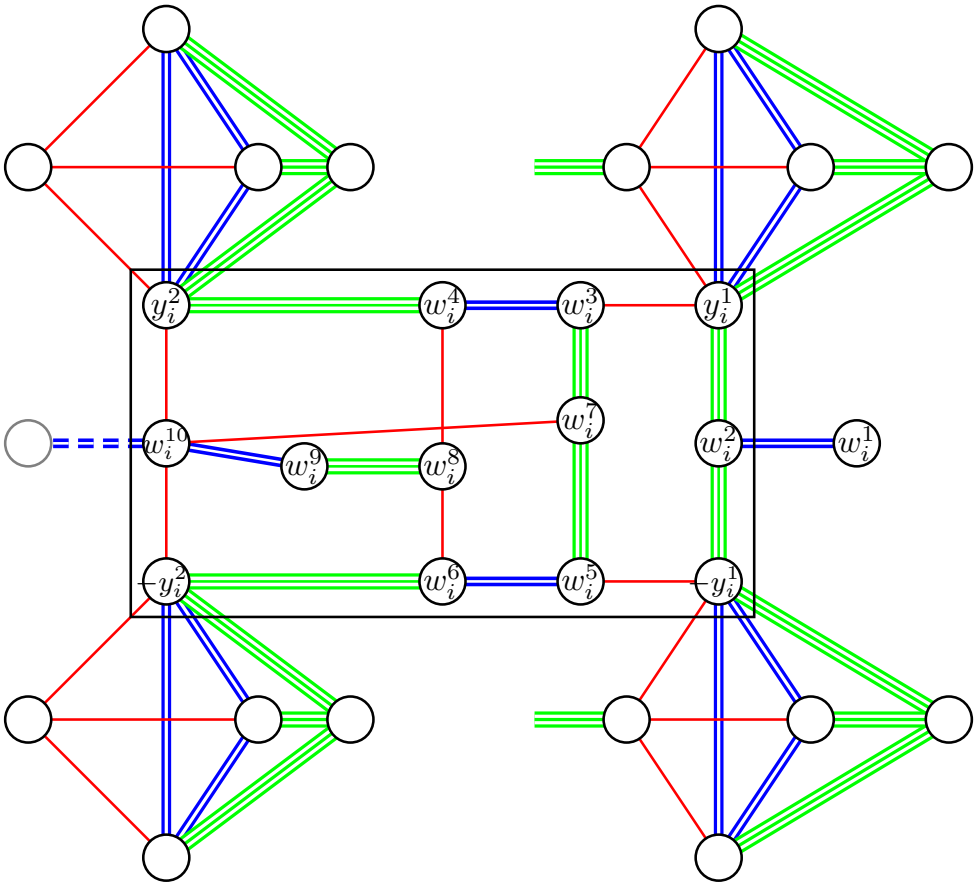



**Figure 3** Unitary variable-gadget (framed). The four adjacent clause-gadgets are depicted too. The meaning of the color is the same as in Figure 2.

### 4.1.3 The trap

With the variable- and clause-gadgets, the snapshots are not connected. We make them connected by adding a gadget, called trap, from which it is impossible to go out. Hence, the trap must be explored last.

The trap, depicted in Figure 4, has 3 vertices $A$, $B$ and $C$ and works as follows. On times $k \equiv 1 \bmod 3$ (resp. $k \equiv 2 \bmod 3$, $k \equiv 0 \bmod 3$), all vertices from the rest of the graphs are adjacent to $A$ (resp. $B$, $C$). If an agent enters the trap, it will alternate between vertices $A$, $B$ and $C$ and cannot go out.

### 4.1.4 Validity of the reduction

We are now ready to prove that the formula is satisfiable if and only if the graph can be fully explored, starting at the vertex $v_1^1$ (gadget of the first clause).

▶ **Lemma 13.** *If the formula is satisfiable then the agent can explore the whole graph.*

**Proof.** Consider a truth assignment $\sigma$ satisfying the formula. We consider the following exploration of the graph. The agent visits first the clause-gadgets, from the first to the last one. As explained earlier, they can do this while exploring all but one vertex among

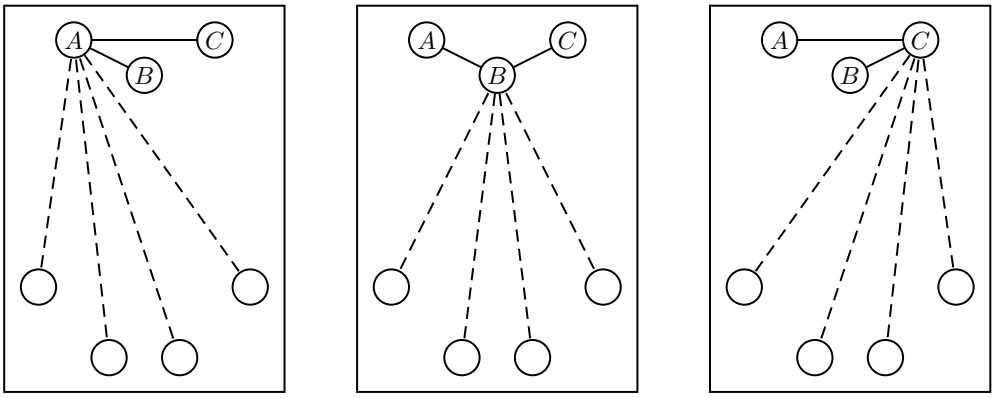



**Figure 4** The three different snapshots of the trap-gadget.

$\{\ell_i^1, \ell_i^2, \ell_i^3\}$ in each clause-gadget. The vertex we choose *not* to visit is one that corresponds to a satisfied literal in $\sigma$.

We are at $v_m^5$ at some time $t \equiv 1 \bmod 3$. We then enter the first variable-gadget at $t+1 \equiv 2 \bmod 3$. Then we visit all the variable-gadgets using path $P_1$ if $y_i$ is set to true, using path $P_2$ if $y_i$ is set to false. After visiting all the variable-gadgets, we go to the trap and visit $A$, $B$ and $C$.

As in the clause-gadget the vertex which was not visited during the exploration of the gadget is set to true, it is visited during the exploration of the corresponding variable-gadget. Thus, all the vertices of the graph are visited. ◀

▶ **Lemma 14.** *If the agent can explore the whole graph then the formula is satisfiable.*

**Proof.** The agent starts at $v_1^1$ at time 1. As noted previously, the trap must be visited at the end of the exploration.

We look at the first vertex $w_1^1$ of the first variable-gadget. The agent is necessarily here at some time $t \equiv 2 \bmod 3$ (from the edge $(v_m^5, w_1^1)$, as the agent cannot be at $w_1^2$ at time $\equiv 2 \bmod 3$, so the edge $(w_1^2, w_1^1)$ cannot be used to reach $w_1^1$). Then, as mentioned earlier, in this variable-gadget they are in $y_1^j$ or $-y_1^j$ only at time $\equiv 1 \bmod 3$. Suppose that they leave the variable-gadget and enters a clause gadget. Then they must take an edge leading to a first vertex $v_i^1$ of the clause gadget, and the agent is stuck there (i.e., they must go to the trap and cannot finish the exploration). So, the agent must follow either path $P_1$ or path $P_2$. They are in $w_1^{10}$, i.e. in the first vertex of the second variable-gadget at some time $\equiv 2 \bmod 3$. More generally, this means that when the agent reaches the first vertex $w_i^1$ of a variable-gadget, then they must stay inside the variable-gadgets, and visit all the subsequent such gadgets, using $P_1$ or $P_2$, till the last vertex $w_n^{10}$ of the last gadget. Then, they must go to the trap.

The exploration starts from vertex $v_1^1$. Suppose now that the exploration leaves some clause-gadget (before $v_m^5$). This must be at some $\ell_i^j$, where they are either at time 2 or 3 mod 3 (coming from $v_i^1$ or some $\ell_i^{j'}$). There is no edge present at time 2 mod 3 incident at some $y_i$-vertex in the variable-gadget.

- If $\ell_i^j$ is a vertex $y_i^1$ or $-y_i^1$, then, to leave the clause-gadget, the agent must reach $w_i^2$ at time 1 mod 3, and is stuck there.
- If $\ell_i^j$ is a vertex $y_i^2$ (resp. $-y_i^2$), then, to leave the clause-gadget, the agent must reach $w_i^4$ (resp. $w_i^6$), then $w_i^8$ at time 2 mod 3, and is stuck here.

In summary, the agent must first visit the clause-gadgets without leaving them. So, they arrive at $v_m^5$ while having visited all the corresponding vertices but one in each clause-gadget. Then, they must follow one path among $P_1$ and $P_2$ in each variable-gadget, and the visit the trap.

We set variable $y_i$ to true (resp. to false) if the agent took path $P_1$ (resp. $P_2$) when visiting the corresponding variable-gadget. As all the vertices have been explored, every $\ell_i^j$ left unexplored in the visit of the clause-gadgets corresponds to a literal set to true in the truth assignment, which concludes the proof.

◀

## 4.2 Case $p \geq 4$

We now show Theorem 3 for $p \geq 4$. The proof is based on a similar reduction from 3-Sat-(2,2). We first extend the clause-gadget and the trap in Section 4.2.1. We then build the variable-gadget in Section 4.2.2. We prove the validity of the reduction in Section 4.2.3.

### 4.2.1 Extending the clause-gadget and the trap

In order to extend the clause-gadget to a larger period, one has to extend its tail. More precisely, we start with the same gadget as for period 3 (up to the fact that edges are present on time modulo $p$ instead of modulo 3). We add $p-3$ vertices $v_i^6, \ldots, v_i^{p+2}$. For $j = 6, \ldots, p+2$, vertex $v_i^{j-1}$ is linked to $v_i^j$ with an edge present on times $k \equiv j-2 \bmod p$. Now, vertex $v_i^{p+2}$ of the gadget of clause $C_i$ is merged with vertex $v_{i+1}^1$ of the gadget of clause $C_{i+1}$.

It is easy to see that the gadget works the same as in the case of period 3: if the agent is at $v_1^1$ at time 1 modulo $p$ and stays inside the clause-gadgets, then they will be at $v_m^{p+2}$ at time 1 modulo $p$, and will have visited every vertex but one $\ell_i^j$ for every $i$.

The trap is generalized in a natural way: it has $p$ vertices, say $f_1, \ldots, f_p$. At time $i$ modulo $p$, $f_i$ is linked to all other vertices of the graph (including the other vertices of the trap). Then, if an agent enters the trap at time $i$ modulo $p$ (in $f_i$), at $i+1$ they must go to $f_{i+1}$, and so on. They can visit all the vertices of the trap but never get out of it.

### 4.2.2 Variable-gadget

The variable-gadget is simpler for $p \geq 4$. As previously, to each variable $y_i$ are already associated 4 vertices in the clause-gadgets, two associated to the literal $y_i$ (denoted $y_i^1$ and $y_i^2$ here) and two associated to the literal $\overline{y_i}$ (denoted $-y_i^1$ and $-y_i^2$ here). The gadget contains $p-1$ vertices $w_i^1, \ldots, w_i^{p-1}$, and the following edges:

- One edge $(w_i^1, w_i^2)$ present on times $k \equiv 2 \bmod p$;
- Two edges $(w_i^2, y_i^1)$ and $(w_i^2, -y_i^1)$ present on times $k \equiv 3 \bmod p$;
- Two edges $(y_i^1, y_i^2)$ and $(-y_i^1, -y_i^2)$ present on times $k \equiv 4 \bmod p$;
- Two edges $(y_i^2, w_i^3)$ and $(-y_i^2, w_i^3)$ present on times $k \equiv 5 \bmod p$;
- If $p \geq 5$, for each $j = 3, \ldots, p-2$, an edge $(w_i^j, w_i^{j+1})$ present on times $k \equiv j+3$.

This construction is illustrated in Figure 5.

Let us call $P_1$ the path $(w_i^1, w_i^2, y_i^1, y_i^2, w_i^3, \ldots, w_i^{p-1})$ and $P_2$ the path $(w_i^1, w_i^2, -y_i^1, -y_i^2, w_i^3, \ldots, w_i^{p-1})$. Note that if an agent is at $w_i^1$ at time 2 mod $p$, then they can follow either $P_1$ or $P_2$, and be at $w_i^{p-1}$ at time 2 mod $p$. If they stay inside the gadget, then these are the only 2 paths that they may follow.

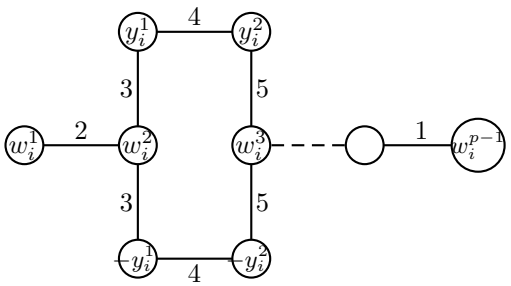



**Figure 5** The variable-gadget. The number above each edge denotes the timeframes where it appears.

To complete the construction, the last vertex $w_i^{p-1}$ of a variable-gadget is merged with the first vertex $w_{i+1}^1$ of the next variable-gadget. Also, there is an edge, present at times 1 mod $p$, between the last vertex $v_m^{p+2}$ of the last clause-gadget and the first vertex $w_1^1$ of the first variable-gadget.

### 4.2.3 Validity of the reduction

We consider the exploration problem starting at vertex $v_i^1$.

▶ **Lemma 15.** *An agent can explore the whole graph if and only if the formula is satisfiable.*

**Proof.** If the formula is satisfiable, let us consider a satisfying assignment $\sigma$. As in the case of period 3, an agent can first visit the clause-gadgets, visiting all vertices but one per clause-gadget which corresponds to a true literal in $\sigma$. They arrive in $v_m^{p+2}$ at time 1 modulo $p$. Then they go to the first vertex of the first variable-gadget, and visit all the variable-gadgets, choosing path $P_1$ if the variable is true in $\sigma$, and $P_2$ otherwise. Finally, they go to the trap and visits it. With the very same argument as previously, we see that they have visited all the vertices of the graph.

Conversely, suppose that an agent can explore all the vertices. Suppose first that they leave a clause-gadget, at some vertex in $\{y_i^1, y_i^2, -y_i^1, -y_i^2\}$. They must be in this vertex at time 2 or 3 modulo $p$.

- If it is $y_i^1$ or $-y_i^1$, they are stuck if they are there at time 2. If they are there at time 3, they can go to $w_i^2$, but then they are stuck there.
- If it is $y_i^2$ or $-y_i^2$, they are immediately stuck.

So, the agent must go through all the clause-gadget first, till $v_m^{p+2}$. Similarly, it is easy to see that an agent cannot leave a variable-gadget:

- they are at vertices $y_i^1$ or $-y_i^1$ at time 4 modulo $p$, and cannot use an edge of the clause-gadget then,
- they are at vertices $y_i^2$ or $-y_i^2$ at time 5 modulo $p$. If $p \geq 5$ they are stuck there, and if $p = 4$, they can go to the first vertex $v_i^1$ of some clause-gadget but then are stuck there.

To conclude, they have to visit first all the clause-gadgets, then all the variable-gadgets, then the trap. We set variable $y_i$ to true (resp. false) if they used path $P_1$ (resp. $P_2$) in the corresponding variable-gadget. As in the case of period 3, from the fact that the agent visits all the vertices of the clause-gadgets, we conclude that every clause has a true literal in the built assignment. ◀

## 5 Exploration time bounds

In this section we focus on the time required to explore the temporal graph (whenever possible). Recall that in the model where the agent is not forced to move, time $O(n^2)$ is sufficient to explore a temporal graph with connected snapshots, and that this bound is tight ($\Omega(n^2)$ steps are necessary for some family of temporal graphs).

We prove here similar results for $p$-periodical temporal graphs under the restlessness 1-constraint.

▶ **Theorem 16.** *Any (restlessly) fully explorable $p$-periodical temporal graph can be explored in at most $pn^2$ steps. Moreover, for every $p \geq 2$ there are families of fully explorable $p$-periodical temporal graphs that require $\Omega(pn^2)$ steps to be fully explored.*

We note that the lower bound $\Omega(pn^2)$ is still valid under the condition that the snapshots are connected (using a trap-structure as in the NP-hardness proof, we can easily make them connected without changing significantly the exploration time bound).

The first part of the theorem, restated in the following lemma, easily follows from Lemma 7.

▶ **Lemma 17.** *Any explorable $p$-periodical temporal graph can be explored restlessly in at most $pn^2$ steps.*

**Proof.** If the graph is explorable, let us consider a journey that visits all vertices. By Lemma 7, we can go from one vertex to the next one in the walk with a walk of at most $pn - 1$ edges. The result follows. ◀

Let us now show the second part of the theorem, restated in the following lemma.

▶ **Lemma 18.** *For every $p \geq 2$ there are families of explorable $p$-periodical temporal graphs that require $\Omega(pn^2)$ steps to be explored.*

**Proof.** We first consider the case where $p$ is even. We build the following graph: first, we consider two even cycles $C_1 = (s, v_2, \ldots, v_{2k}, s)$ and $C_2 = (s, z_2, \ldots, z_{2k}, s)$ of the same size $2k$, sharing a vertex $s$. We choose $k$ (larger than $p$) such that $2k \equiv 2 \bmod p$. In $C_1$, $(s, v_2)$ and every $(v_{2i-1}, v_{2i})$ are present at odd times, and the other edges (including $(v_{2k}, s)$) are present at even times. In $C_2$, starting from $s$ through $z_2$, $z_3$,..., the $i^{\text{th}}$ edge is present at times $\equiv i \bmod p$.

To complete the construction, we split vertex $z_2$ into $2k+1$ identical copies (so each copy is linked to $s$ and to $z_3$). The initial vertex is $s$. Note that the graph has $6k - 1$ vertices.

This consutrction is illustrated in Figure 6

Let us suppose that at some time $t \equiv 1 \bmod p$, the agent is in $s$ and enters $C_2$. Then they must follow the whole cycle, visiting one copy of $z_2$, then $z_3$, ..., $z_{2k}$, being back in $s$ at $t' = t + 2k \equiv 3 \bmod p$. Then, if $p \neq 2$ they must follow $C_1$ (visiting $v_2$, $v_3$,...), being back in $s$ at $t'' = t' + 2k \equiv 5 \bmod p$. In order to be able to go back in $C_2$ again, they must make $p/2 - 1$ tours of $C_1$, so that they are in $s$ at some time $\equiv 1 \bmod p$ (note that this is also true for $p = 2$ as $p/2 - 1 = 0$).

Then, starting at $s$ at time 1, as the agent must go $2k + 1$ times through $C_2$ in order to visit all copies of $z_2$, they must make (at least) $2k$ (complete) tours of $C_2$ and $2k(p/2 - 1)$ (complete) tours of $C_1$, so in total at least $kp$ tours of cycles of length $2k$, leading to an exploration time at least $2k^2p \geq n^2p/18 = \Omega(n^2p)$.

It is easy to see that the graph is indeed explorable with the previous strategy.

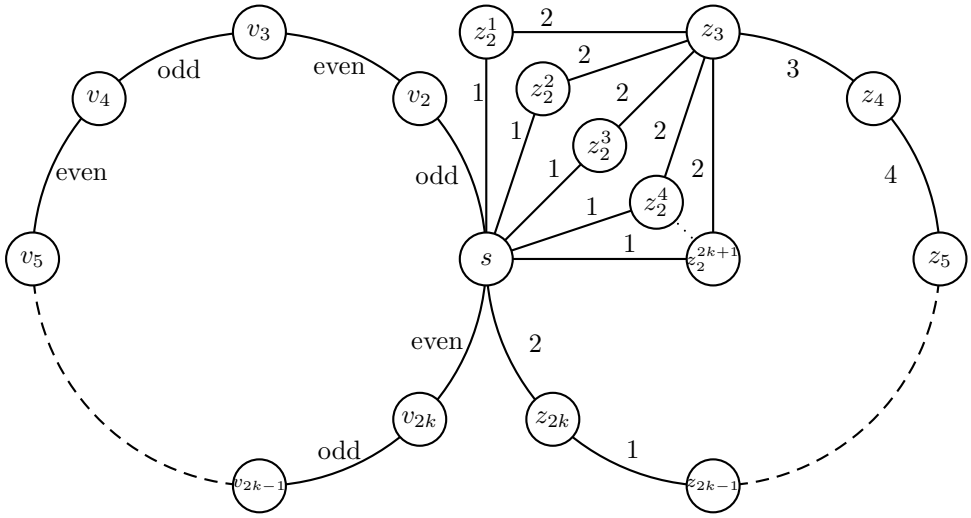



**Figure 6** The graph our construction provides. $C_1$ is on the left, $C_2$ on the right.

If the period is odd, it is no longer possible for an edge to only appear at odd or at even timeframes and our construction requires a little adaptation. We modify $C_1$ as follows:

- Edges $(s, v_2)$ and $(v_{2i-1}, v_{2i})$ are present at times equivalent to $1, 3, \ldots, p-2 \bmod p$, let us call them odd edges;
- Edges $(v_{2i}, v_{2i+1})$ and $(v_{2k}, s)$ are present at times equivalent to $2, 4, \ldots, p-1 \bmod p$, let us call them even edges;
- For each even edge, we add a vertex $w_i$, with an edge $(v_{2i}, w_i)$ present at $p-1 \bmod p$, and an edge $(w_i, v_{2i+1})$ (or $(w_i, s)$) present at $0 \bmod p$.

For example, Figure 7 illustrates the changes we make to the graph of Figure 6 if $p$ is odd.

This way, if the agent is in $C_1$, they will alternate between odd and even edges, up to time $p-1 \bmod p$, where they are at an "odd" vertex $v_{2i}$, has to use the extra vertex $w_i$, leaving $w_i$ at $0 \bmod p$, reaching $v_{2i+1}$ (or $s$ if $i = k$) at time $1 \bmod p$. Then, in $p$ units of times they travel from $v_i$ to $v_{i+p-1}$. We set the length of the cycle $C_1$ (on $s$ and the vertices $v_i$) to be $2k \equiv 2 \bmod (p-1)$.

Then in this configuration, if the agent starts visiting the tour $C_1$ at $s \equiv 3 \bmod p$, it ends the tour (visiting some vertices $w$ during the travel) at $s$ at time $t' \equiv 5 \bmod p$, and so on. After $(p-3)/2$ such tours, it will be at $s$ at $t \equiv 0 \bmod p$, will start a new tour of $C_1$, will be at $v_{2k}$ at time $t' \equiv 0 \bmod p$, and then can use the edge $(v_{2k}, s)$ to be back on $s$ at time $\equiv 1 \bmod p$.

So, to visit all copies of $z_2$, the agent must make at least $2k$ (complete) tours of $C_2$, and $2k(p-1)/2$ tours of $C_1$. As tours have (at least) $2k$ vertices, we get again a lower bound of $2k^2p = \Omega(pn^2)$ to explore the whole graph restlessly.

Here again, the previous strategy shows that the graph is fully explorable. ◀

## 6 Conclusion and perspective

In the previous section, we proved that the maximum amount of time required for the exploration of an explorable $p$-periodic graph of $n$ vertices is in $\Theta(pn^2)$. However, we only proved that this value is somewhere between $\frac{pn^2}{18}$ and $pn^2$ and we leave its exact value as an open question.

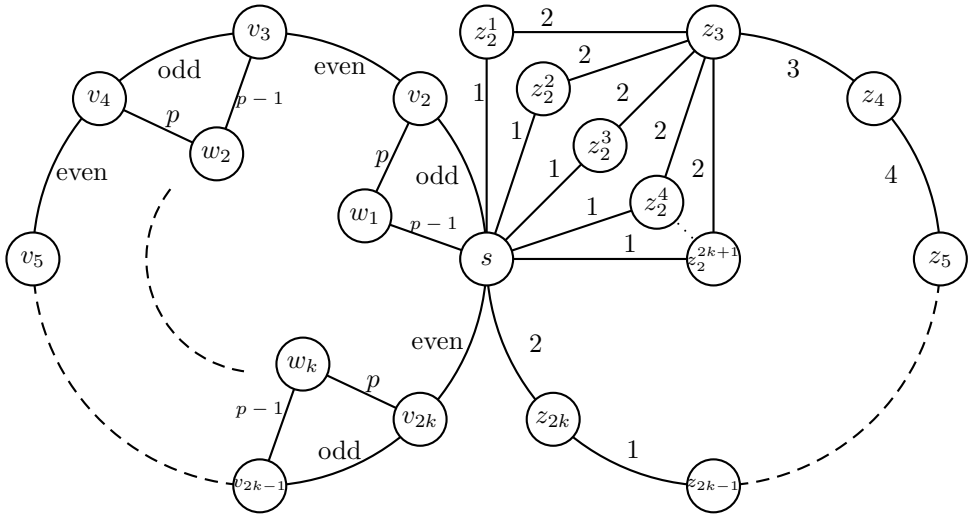



**Figure 7** We add an extra vertex $w_i$ for every odd edge. Note that here, even edges are present on timeframes $2, 4, ..., p-1$ and odd edges on timeframes $1, 3, 5, ..., p-2$ but not $p$, which is dealt with separately.

Another natural extensions of our work could be to investigate the complexity of $\Delta$-restless exploration with $\Delta > 1$ or cases where the agent walking in the graph is allowed to move by more than one edge at each time step but we do not know if those cases would be more interesting than the case $\Delta = 1$.

A problem that we believe would be of great interest is the study of the tractability of the NP-complete cases of the problem for some fixed relevant parameters. This could include structural parameters of the underlying graph (the timeless graph that contains all the edges that appear at any given time) or parameters related to the temporality. In particular, our gadgets were inspired by our previous works on edge-colored graphs and most edges are present in only one snapshot, which makes the graph change drastically from a timeframe to the next. We believe it could be of great interest to study the complexity of cases where the graph cannot change too much in one timeframe.

## References


1 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs – Theory, Algorithms and Applications, Second Edition.* Springer Monographs in Mathematics. Springer, 2009.

2 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electron. Colloquium Comput. Complex.*, TR03-049, 2003. `arXiv:TR03-049`.

3 Hans L. Bodlaender and Tom C. van der Zanden. On exploring always-connected temporal graphs of small pathwidth. *Inf. Process. Lett.*, 142:68–71, 2019. `doi:10.1016/j.ipl.2018.10.016`.

4 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. `doi:10.1007/s00453-021-00831-w`.

5 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. `doi:10.1016/j.jcss.2021.01.005`.

6 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132

of *LIPIcs*, pages 141:1–141:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.141.

7 Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.15.

8 Petter Holme. Temporal network structures controlling disease spreading. *Phys. Rev. E*, 94:022305, August 2016. doi:10.1103/PhysRevE.94.022305.

9 David Ilcinkas, Ralf Klasing, and Ahmed Mouhamadou Wade. Exploration of constantly connected dynamic graphs based on cactuses. In Magnús M. Halldórsson, editor, *Structural Information and Communication Complexity – 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, volume 8576 of *Lecture Notes in Computer Science*, pages 250–262. Springer, 2014. doi:10.1007/978-3-319-09620-9_20.

10 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.

**Revision Notice**

This is a revised version of the eponymous paper appeared in the proceedings of SAND 2023 (LIPIcs, volume 257, https://www.dagstuhl.de/dagpub/978-3-95977-275-4, published in June, 2023), in which the variable-gadget construction of Section 4.1.2 has been changed (in particular the gadget depicted in Figure 3), to correct a flaw in the initial version.

*Dagstuhl Publishing – July 5, 2023.*

# Multistage Shortest Path: Instances and Practical Evaluation

**Markus Chimani**
Theoretical Computer Science, Universität Osnabrück, Germany

**Niklas Troost**
Theoretical Computer Science, Universität Osnabrück, Germany

## Abstract

A multistage graph problem is a generalization of a traditional graph problem where, instead of a single input graph, we consider a sequence of graphs. We ask for a sequence of solutions, one for each input graph, such that consecutive solutions are as similar as possible. There are several theoretical results on different multistage problems and their complexities, as well as FPT and approximation algorithms. However, there is a severe lack of experimental validation and resulting feedback. Not only are there no algorithmic experiments in literature, we do not even know of any strong set of multistage benchmark instances.

In this paper we want to improve on this situation. We consider the natural problem of multistage shortest path (MSP). First, we propose a rich benchmark set, ranging from synthetic to real-world data, and discuss relevant aspects to ensure non-trivial instances, which is a surprisingly delicate task. Secondly, we present an explorative study on heuristic, approximate, and exact algorithms for the MSP problem from a practical point of view. Our practical findings also inform theoretical research in arguing sensible further directions. For example, based on our study we propose to focus on algorithms for multistage instances that do not rely on 2-stage oracles.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Multistage Graphs, Shortest Paths, Experiments

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2023.14

**Supplementary Material** *Dataset (Benchmark Instances)*: `https://tcs.uos.de/research/msp`

## 1 Introduction

In multistage problems, as introduced in their current form by [13, 18], we are interested in solving some problem not on a single instance, but on a sequence of instances (the *stages*) which correspond to different points in time. Such problems arise, e.g., when a certain task has to be performed multiple times at discrete points in time, but the underlying instance (in our case a graph) has received several modifications between two such time points. Thus, one typically expects two succeeding stages to be somewhat similar overall, but certainly more significantly different than being attained from a single graph operation like adding or deleting an edge. Most importantly, additionally to the original problem's objective per stage (the *stage-wise objective*), we also aim to maximize the similarity between the individual stage's solutions – the *transition quality*.

Having two distinct optimization goals at hand, one typically considers a weighted sum of both measures to allow trade-offs between the quality of the individual solutions and the similarity of those solutions [1–4, 15–17, 19]. Sometimes it is desired to guarantee optimal solutions in each stage, as first motivated in [8]; then the goal is to maximize the transition qualities by picking a suitable optimal solution (out of the set of possible optimal solutions) per stage. In either case, one may not want to yield the largest possible transition quality between two stages if this incurred an exorbitant quality decrease in other stage transitions.


© Markus Chimani and Niklas Troost;
licensed under Creative Commons License CC-BY 4.0

Also note that our problem notion is different from many other scenarios on dynamic graphs, where the goal is to – possibly after each graph modification – update a solution as fast as possible, not (directly) caring about the specific amount of changes to the solution.

Interestingly, most polynomial-time solvable graph problems (such as shortest paths, matchings, minimum cuts, etc.) yield NP-complete problems in a multistage setting: this often already occurs when only two stages are considered, and independent on whether one restricts themselves to optimal solutions per stage or not [7, 19]. There is already some theoretical research on several variants of this problem framework; however, there is significant lack of practical evaluation. In fact, it seems that there have been no practical evaluations on any multistage graph problem so far. In this paper, we want to improve on this situation.

To this end, we consider the Multistage Shortest Path (MSP) problem, which seems to probably be the most practically relevant multistage problem. MSP was first proposed in [18] and introduced with a trade-off objective in [17]. We discuss it here in the setting where we only allow optimal solutions per stage: Given an ordered set of edge-weighted graphs (the stages) and a node pair $(s, t)$, find a shortest $s$-$t$-path in each stage such that the subsequent paths are as similar as possible (see Section 2 for a formal definition). For example in a transportation scenario, it might be necessary but expensive to prepare each road segment before using it. Thus, we want a collection of shortest paths that allows us to reuse as many segments as possible. In a communication scenario, we prefer to use recently established channels, but not at the cost of sacrificing transfer speed. If the optimality requirement per stage appears too restricting, we point out that one can easily relax it in practice by altering the notion of what a *shortest* path is, e.g., by rounding edge weights so that all paths of reasonably similar length are considered optimal.

While the usual shortest $s$-$t$-path problem is long known to be efficiently solvable using Dijkstra's algorithm [11], MSP was shown to be NP-hard even for unweighted instances via a reduction from 3Sat [17]. Although not stated explicitly, the proof can easily be adapted to an approximation-preserving reduction from Max-2Sat which shows that, unless P = NP, MSP does not admit a PTAS nor a constant-factor approximation with factor better than 21/22 [20], even when restricted to only two stages. In [17], several similarity and dissimilarity measures were considered, and several results w.r.t. the parameterized complexity of MSP could be established. The specific formulation above, with only truly shortest paths per stage, is motivated by [8] and explicitly considered in [7] in the context of approximation algorithms.

**Contribution.** In this paper we improve on the state-of-the-art regarding practical algorithmics in the following two ways: First (Section 3), we propose the first rich benchmark sets for a multistage graph problem. We take special care to avoid ad-hoc generation schemes and parameterizations which, in case of MSP, would typically only yield rather trivial instances. Instead, we devise several explicit measures of reasons for triviality and actively seek schemes and parameterizations to avoid them. Secondly, we implemented and tested a set of heuristic, approximate, and exact algorithms to tackle MSP in practice (Section 4), and we report on our explorative study (Section 5). A focus of this study is to test the consistency between theoretical results and their practical realization and use it as a source for identifying new research questions.

Theoretical research suggests to improve on algorithms for the (formally already hard) 2-stage problem variant $\mathsf{MSP}|_2$, as we only know a single approximation algorithm (with non-constant approximation ratio). The multistage variants with more than two stages can reuse any 2-stage algorithm while weakening its approximation ratio only by the constant

ratio of $1/2$. Interestingly, we find that in practice $\mathsf{MSP}|_2$ problems are all rather simple to solve, but neither the known approximation nor other heuristics yield satisfactory results for general MSP. Thus, we propose that a promising step for theoretical research would be to further investigate the intricacies of the true multistage setting instead of relying on algorithms for a small constant number of stages.

The implementations will be part of the next release of the open-source (GPL) *Open Graph algorithms and Data structures Framework* [6] (`www.ogdf.net`); all benchmark instances and experimental data are available at `https://tcs.uos.de/research/msp`.

## 2 Definitions and Preliminaries

Given a graph $G$ with positive edge weights $w\colon E(G) \to \mathbb{R}^+$, we encode a path $P \subseteq E(G)$ as an edge set and denote its *path length* by $\ell(P) \coloneqq \sum_{e\in P} w(e)$. Given a *query* $(s,t) \in V(G)^2$, a *shortest $s$-$t$-path* is an $s$-$t$-path with minimum path length. In contrast, we may also consider the number of *hops* (edges) $|P|$ of a path $P$. The *hop-distance* $h(s,t)$ is the smallest number of hops over all $s$-$t$-paths.

Let $[k] \coloneqq \{1, 2, ..., k\}$. We define a multistage graph[1] $\mathcal{G}^\tau = \langle G_i, w_i\rangle_{i\in[\tau]}$ as an ordered sequence of graphs with positive edge weights over a common node set $V$, i.e., $G_i = (V, E_i)$ and $w_i\colon E_i \to \mathbb{R}^+$ for all $i \in [\tau]$. Each tuple $(G_i, w_i)$ is a *stage*, and $\mathcal{G}^\tau$ has $\tau$ stages. Observe that the weights of common edges may differ between stages.

▶ **Definition 1** (Multistage Shortest Path (MSP)). *Given a multistage graph $\mathcal{G}^\tau$ and a query $(s,t) \in V^2$, find a sequence $\mathcal{P} \coloneqq \langle P_i\rangle_{i\in[\tau]}$ of paths such that each $P_i$ is a shortest $s$-$t$-path in $G_i$ and the transition quality $Q(\mathcal{P}) \coloneqq \sum_{i\in[\tau-1]} |P_i \cap P_{i+1}|$ is maximized.*

If there is an upper bound $T$ on the number of stages $\tau$, MSP may be denoted by $\mathsf{MSP}|_T$.

As the problem is NP-hard, we may be interested in approximate solutions. The only known approximation algorithms for $\mathsf{MSP}|_2$ and general MSP arise as special cases of a general approximation framework [7], which in turn is a generalization of the approximation for multistage matching [8]. We will briefly summarize the algorithms later in Sections 4.2 and 4.3. For now, we may only mention that the approximation ratio is dependent on the *intertwinement* $\mu \coloneqq \max_{i\in[\tau-1]} |E_i \cap E_{i+1}|$ of the multistage graph, i.e., the maximum commonality between the edge sets of two succeeding stages. For $\mathsf{MSP}|_2$ and MSP the algorithms yield approximation ratios of $(2\mu)^{-1/2}$ and $(8\mu)^{-1/2}$, respectively. While [7] guarantees these ratios, their tightness cannot be deduced for arbitrary subgraph problems. However, following the construction ideas of [8], it is simple to show the tightness of the ratio (up to a small constant) for MSP and $\mathsf{MSP}|_2$.

**Preprocessing.** For a given query $(s,t) \in V^2$ and looking at any stage individually, we may remove all its *non-essential* edges (i.e., edges that are not in any shortest $s$-$t$-path in that stage) without altering the set of feasible solutions. We may also discard (arising) degree-0 nodes. This can be done efficiently, as given in Algorithm 1. Thus, we assume in the following that this preprocessing is always performed before running the actual algorithms. A stage $G_i$ that has been preprocessed w.r.t. a query $(s,t)$ has the following useful properties:

- **(i)** $G_i$ is a DAG with unique root $s$ and unique sink $t$, and
- **(ii)** for each node $v \in V(G_i)$, all paths from $s$ to $v$ have the same length. The same holds for all paths from $v$ to $t$.

[1] This term is also sometimes used for *leveled* graphs, whose nodes are partitioned into levels, and edges join consecutive levels, see, e.g., [10]. That definition and results thereon are unrelated to our scenario.

**Algorithm 1** **Preprocessing** non-essential elements in an edge-weighted graph $G=(V,E)$.

1 compute shortest path distances $d(v)$ from $s$ to each $v \in V$ using Dijkstra's algorithm
2 remove all edges $\{(u,v) \in E \mid d(u) + w(u,v) \neq d(v)\}$
3 compute all nodes $U$ with a path to $t$ (via BFS from $t$ with reversed edges)
4 remove all nodes $V \setminus U$

After the stage-wise preprocessing, both properties in particular also hold for the graph induced by the intersection $E_i \cap E_{i+1}$, for each $i \in [\tau - 1]$.

# 3 Benchmark Instances

Multistage problems have mostly been viewed from a theoretical perspective up to now, and there are thus no established sets of *stage-wise* temporal instances available. Furthermore, it turns out that acquiring and even generating reasonable instances is no easy feat: In our investigations, we learned that most ad-hoc generation schemes typically lead to rather trivial multistage instances. If there are only very few different (or even just one unique) shortest paths per stage, there is not much room for transition optimization; if there are several shortest paths but on very similar stages, chances are that a single solution path can be chosen throughout all stages; if the stages become too dissimilar, such that they have only few edges in common between shortest paths, it again becomes rather simple to select shortest paths that agree in terms of these edges between subsequent stages.

An adversary may argue that such issues would go away if one switches to a trade-off based objective function where the paths' lengths are allowed to deviate from the optimum in order to allow better transitions. But we disagree: Generating instances with only a trade-off based objective function in mind would easily hide the fact that such instances may become trivial for different balancing ratios between the two considered objective functions. If, however, the instances are well-designed for our scenario with truly optimal shortest paths, we expect them to be also interesting for trade-off based optimization. Thus, it is important to discuss our benchmark generating procedure in more detail than is often done otherwise. While we cannot guarantee that our instances are especially sensible for problems beyond MSP, we hope that the underlying generation methods and considerations may be useful for devising new instances for experimental studies on other multistage problems as well.

We consider four different types of benchmark instances, each with slightly different focus and motivation (see Section 3.1), and ranging from highly synthetic to real-world origins. After discussing schemes of obtaining MSP instances from underlying graphs in Section 3.2, we discuss the complexities of identifying good parameters to obtain non-trivial MSP instances in Section 3.3. Then, Section 3.4 presents the final technical parameterizations of our benchmark sets and resulting instance properties.

## 3.1 Rationale for the Benchmark Scenarios

As discussed in Section 2, the query-specific preprocessing of MSP instances may yield vastly smaller stages, and the preprocessed stages have a very specific structure. In particular, their size is not necessarily related to the original instance size anymore. To obtain interesting instances, a main goal is thus to generate instances with a reasonable number of shortest paths with a reasonable number of hops each, so that the preprocessing does not already essentially solve the instance.

To this end, we start with generating a highly synthetic benchmark set `grid`, which consists of long grid graphs (i.e., two-dimensional grids where one dimension is significantly smaller than the other). For a query $(s, t)$ where $s$ ($t$) is the lower left (upper right, respectively) corner of the grid, these graphs (assuming unit edge weights) already resemble preprocessed MSP instances. Further, in contrast to more quadratically-shaped grid graphs, even relatively small modifications to the graphs are likely to yield non-trivial instances.

The benchmark set `geom` contains nearest-neighbor graphs [14], generated by a random point set in the Euclidean plane. Such random graphs allow for multiple shortest paths of reasonable lengths. In contrast, other well-established randomized generation paradigms like Erdős-Renyi graphs or Barabási-Albert graphs would only yield very small diameters [5, 9]. Further, our geometric graphs have the additional benefit of (i) naturally occurring edge weights, and (ii) if one stage is generated from the previous stage by adding some random displacement to each node, they also provide a natural temporal relationship between consecutive stages.

The probably most natural application for shortest path queries is navigation in road networks. However, readily available data sets do not include temporal data suitable for MSP. Our benchmark set `hybr` thus uses real-world road networks as the underlying graph data, for which we artificially generate temporal differences between the stages. Our modification methods (see Section 3.2) are mainly motivated by this scenario, but we use the same modification methods for the previous two benchmark sets as well.

Finally, there exist real-world data sets from other applications that include time-stamped edges. Under those, we are mainly interested in email communication networks ("who wrote to whom, and when?") or human contact networks ("who was near whom, and when?"), as we can interpret these data in the context of the MSP problem: We want to quickly pass some information from source to target, while preferring interpersonal relations that have been used recently. We collect such instances in our benchmark set `real`.

## 3.2 Multistage Instance Generation

**Modification variants.** Necessarily, the stages of a multistage graph need to differ to compose a non-trivial instance. The `real` instances already have differing stages; for `grid`, `geom`, and `hybr` base instances we can use either of the following three modification schemes (applied to each stage independently) to obtain multiple differing stages. Additionally, we can also obtain differing stages for the `geom` instances by perturbing the node coordinates between stages to simulate random walks of the nodes (see Section 3.4). Keep in mind that these modifications are performed on the original graph, prior any query knowledge and thus prior to any preprocessing.

**Edge deletion:** Regardless of the interpretation of the instance, there is a plethora of different reasons to motivate the absence of edges in some stages. As simple examples, road closures in road networks or link failures in computer networks can lead to their temporary unavailability. Given a *modification ratio* $\lambda_E \in [0, 1)$, we remove $\lfloor \lambda_E \cdot |E_i| \rfloor$ many edges from the respective stage, chosen uniformly at random.

**Node deletion:** Removing arbitrarily chosen edges might (i) not alter the set of shortest paths too much and (ii) not describe all real-world scenarios too well, as the reason for an edge absence can have some local impact on surrounding edges as well. A simple method to generate local events of slightly larger impact is to remove nodes together with all incident edges – this occurs, e.g., if some road intersection is blocked, or if some server goes offline in a communication network. As above, we use a *modification ratio* $\lambda_V \in [0, 1)$ and remove $\lfloor \lambda_V \cdot |V| \rfloor$ many nodes from the respective stage, chosen uniformly at random.

**Weight scaling:** Some incidents (e.g., construction sites) do not render the respective node or edge completely unusable but rather increase the cost for their usage. This is typically no isolated effect but also affects the neighborhood of said graph element – the closer the proximity, the larger the effect. We model this by selecting a random node $v$ and sorting $E_i$ by hop-distance from $v$; the closer an edge is to $v$, the more we scale up its weight. The following precise parameters were selected subject to the discussion in Section 3.3: the weights of the $\lfloor |E_i|/8 \rfloor$ closest edges are multiplied by a factor of 4; the next $\lfloor |E_i|/4 \rfloor$ edges are multiplied by a factor of 2. Observe that if edges have exponential weights (base 2, see below) they retain this property after the scaling.

**Query selection.** While the `grid` instances are constructed with specific extremal queries in mind, we need to choose queries for the other three benchmark sets. To find multiple queries, each with relatively long shortest paths, we use the following randomized process. Consider some stage $G_i$ with the least number of edges. Let $h'(v, w) \coloneqq h(v, w)$ denote the hop-distance from $v$ to $w$ in $G_i$ if they are in a common component, and $h'(v, w) \coloneqq -1$ otherwise. Let $h^*(v) \coloneqq \max_{u \in V} h'(v, u)$ and $H(v) \coloneqq \{w \in V \mid h'(v, w) \geq 3/4 \cdot h^*(v)\}$ denote a set of distant nodes from $v$. Starting from a random node $c \in V$ in the largest connected component of $G_i$, we choose the source $s$ uniformly at random from $H(c)$. The target $t$ is in turn chosen uniformly at random from $H(s)$. If the query is not feasible for all stages, it is rejected.

## 3.3 Quality Criteria and Triviality Considerations

To differentiate interesting from trivial instances, multiple aspects have to align. These are specifically derived from our view on the MSP-problem but might also be generalized to classify instances for other multistage problems.

**Triviality in a stage.** Simple kinds of trivialities can be pinpointed to a specific stage.

**Few paths:** Let $N$ denote the number of shortest $s$-$t$-paths in a single stage $G_i$. If $N = 0$, node $t$ is not reachable from $s$ and the instance could be split into two independent sub-instances before and after the $i$-th stage. Alternatively, if a practical application would rather incentivize similarity to the last feasible solution, we could simply remove the infeasible stage. Similarly, if $N = 1$, the shortest $s$-$t$-path is unique in stage $i$, and we can split the instance at this stage. The case $N \leq 1$ is trivial to check after preprocessing, since then $E_i$ is either empty or a single path. We may discard instances with such a stage for experimental purposes. For $N \geq 2$, we consider the ratio $|E_i|/h_i(s, t)$ as a measure to estimate the non-triviality of stage $G_i$. Here, $h_i$ is the hop-distance in $G_i$, easily computable during preprocessing.

**Short paths:** We may disregard instances with too small $h_i$, as defined above.

**Triviality in a transition.** Trivialities arising in transitions between stages may be harder to spot. Furthermore, even after understanding how to generate instances with reasonable stage-wise non-triviality, finding generation parameters that yield transition-wise non-trivial instances turned out to be much more fiddly and required more computational effort. E.g., to compute (or estimate) the measures below, we require optimal (or heuristic) solutions. Section 4.1 discusses a method to (in practice) acquire an optimal solution $\langle P_i \rangle_{i \in [\tau]}$ in reasonable enough time by the use of an ILP.

**Small intersection:** If, after preprocessing, the intersection $E' \coloneqq E_i \cap E_{i+1}$ between two consecutive stages is too small or poorly structured (e.g., if $E'$ consists of mostly disconnected edges), all of $E'$ might be in an optimal solution simultaneously, i.e., $E_i \cap E_{i+1} \subseteq P_i \cap P_{i+1}$. In this case, already the simplest greedy algorithm (see Section 4.2) would always find an

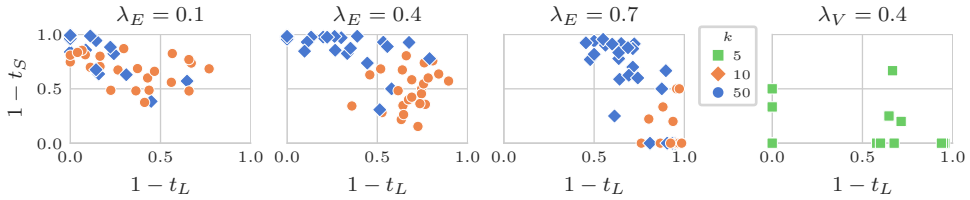



**Figure 1** Triviality measures for 2-stage `geom` instances.

optimal solution. We introduce the triviality measure $t_S := \frac{|P_i \cap P_{i+1}|}{|E_i \cap E_{i+1}|}$ which compares the optimal transition quality to the intersection size. If $t_S = 1$, the transition is trivial; if $t_S$ is close to 0, this triviality aspect plays no important role.

**Large intersection:** If, on the other hand, $E_i \cap E_{i+1}$ is too large, each solution edge may always also be an intersection edge, i.e., $P_i = P_i \cap E_{i+1}$ for any shortest path $P_i$ in $G_i$ (and similarly for $P_{i+1}$). We introduce the triviality measure $t_L := \frac{2 \cdot |P_i \cap P_{i+1}|}{|P_i| + |P_{i+1}|}$, comparing the optimal transition quality with the mean number of hops of the respective shortest paths. If $t_L = 1$, the transition is trivial; if $t_L$ is close to 0, this aspect is not important.

**Small transition quality:** Both triviality measures $t_S$ and $t_L$ are not very expressive if the optimal transition quality $|P_i \cap P_{i+1}|$ is low.

**Identifying non-trivial instances.** The selection of the underlying graphs (and/or their generation methods) allows us to control the non-triviality within single stages in a reasonable and predictable manner. However, controlling the transition-based triviality (mainly $t_S$ and $t_L$) turns out to be much more challenging. This is furthered by the fact that for a nice set of benchmarks, we would like to have similar parameterizations over all instance classes. To this end, we required multiple rounds of generating many instances starting with vastly diverse parameter selections until honing in with fine-grained parameter differences. While this may seem straight-forward on first sight, there are sometimes only very small ranges of suitable parameter values, and they may vastly drift or even disappear by slight changes to other parameters due to the interdependencies of the parameters.

We exemplarily discuss the effects on $t_S$ and $t_L$ by varying the modification parameters for `geom`. See Figure 1 for a visualization, where instances (points in the figure) that are trivial due to a too small (large) intersection tend to the horizontal (vertical, respectively) axis. Consider neighborhood size $k = 10$. For small $\lambda_E$, the intersection is mostly too large, causing low $1 - t_L$; for larger $\lambda_E$, the point set moves down and to the right, rendering more instances to have low $1 - t_S$. This plausible effect is evident for all instance sets, albeit with large discrepancy for sensible values of $\lambda_E$ depending on, for example, $k$: while $\lambda_E = 0.4$ is a sensible choice for $k = 10$, $k = 50$ would benefit from a higher $\lambda_E$ value. Even more so, for some instance parameterizations (e.g., $k = 5$ and $\lambda_V = 0.4$), both $t_S$ and $t_L$ are likely to trigger. Thus, the selected parameters are a compromise between comparability of parameter values and non-triviality w.r.t. all of the above triviality considerations.

## 3.4 Final Parameterization and Generation Details

After multiple rounds of investigations to identify reasonable and consistent parameterizations that yield non-trivial instances, we finally arrive at the following generation settings. Table 1 shows key figures of the generated multistage instances, given as averages over the indicated instance classes. See the appendix for more detailed tables. The full set of benchmark instances, as well as all experimental results, can be found at `https://tcs.uos.de/research/msp`.

**Table 1 Instance characteristics**, grouped by relevant parameters. Here, $n$ and $m$ ($n'$ and $m'$) are the mean number of nodes and edge before (after, respectively) preprocessing, always understood as the union over all stages. The ratios $n'/n$ and $m'/m$ thus measure the effectiveness of the preprocessing strategy. $h$ denotes the mean hop-count of a shortest path per stage; larger numbers typically indicate higher problem difficulty. Value $\mu$ gives the mean intertwinement of the considered instances after preprocessing, which is a measure relevant to the problem's approximability (see Section 2).

| `grid` | $y = 100$ | | $y = 200$ | | $y = 500$ | | $y = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $x$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ |
| 5 | 97.7%<br>97.1% | 362.8<br>105.0 | 98.0%<br>97.3% | 626.1<br>208.6 | 99.2%<br>98.6% | 1396.5<br>519.6 | 99.4%<br>98.9% | 2605.9<br>1037.4 |
| 10 | 89.6%<br>88.0% | 676.5<br>110.2 | 87.8%<br>85.5% | 962.3<br>214.8 | 90.7%<br>87.7% | 1892.5<br>530.6 | 93.9%<br>90.8% | 3476.2<br>1069.0 |
| 25 | 86.9%<br>86.0% | 2356.1<br>123.5 | 77.8%<br>76.1% | 3540.0<br>227.0 | 68.2%<br>64.8% | 4347.4<br>538.6 | 72.0%<br>67.0% | 6834.9<br>1069.2 |
| 50 | 91.1%<br>90.7% | 5506.0<br>148.0 | 80.8%<br>80.1% | 9321.3<br>249.0 | 64.8%<br>63.0% | 14414.3<br>566.4 | 55.4%<br>52.1% | 13940.5<br>1106.7 |

| `geom` | $n = 1000$ | | $n = 2000$ | | $n = 5000$ | |
|---|---|---|---|---|---|---|
| $k$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ | $\frac{n'}{n}$<br>$\frac{m'}{m}$ | $\mu$<br>$h$ |
| 5 | 21.6%<br>10.5% | 53.6<br>30.8 | 17.1%<br>8.4% | 79.0<br>47.2 | 13.7%<br>6.9% | 141.4<br>74.1 |
| 10 | 20.5%<br>7.7% | 60.7<br>20.7 | 18.1%<br>6.6% | 100.6<br>30.9 | 15.0%<br>5.7% | 213.3<br>54.3 |
| 25 | 18.1%<br>4.7% | 120.6<br>12.4 | 17.2%<br>4.5% | 228.1<br>18.5 | 15.5%<br>4.0% | 505.0<br>35.7 |
| 50 | 13.1%<br>2.3% | 165.5<br>7.3 | 15.1%<br>3.0% | 335.2<br>13.3 | 15.6%<br>3.2% | 988.0<br>20.3 |

| `hybr` | $n$ | $m$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ |
|---|---|---|---|---|---|---|
| `CA` | 2.0M | 2.8M | 0.11% | 0.09% | 571.3 | 747.7 |
| `PA` | 1.1M | 1.5M | 0.19% | 0.15% | 551.4 | 653.6 |
| `TX` | 1.4M | 1.9M | 0.19% | 0.15% | 654.0 | 860.0 |

| `real` | $n$ | $m$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ |
|---|---|---|---|---|---|---|
| `dnc.24` | 401.8 | 1.2K | 6.9% | 5.8% | 11.5 | 6.3 |
| `dnc.48` | 536.4 | 1.7K | 5.5% | 5.1% | 15.0 | 5.4 |
| `enron.168` | 5.3K | 12.4K | 1.0% | 0.8% | 11.1 | 7.7 |
| `enron.744` | 8.9K | 25.2K | 0.5% | 0.4% | 12.4 | 7.4 |

**`grid`: Long Grid Graphs.** The underlying grids have $|V| = x \cdot y$ for $(x, y) \in \{5, 10, 25, 50\} \times \{100, 200, 500, 1000\}$ and unit edge weights. For each of the three modification variants we generate MSP instances with 16 stages; each stage is derived from the original underlying graph. We consider the modification ratios $\lambda_E \in \{1/5, 1/10, 1/20\}$ (for $x = 5$, $\lambda_E = 1/5$ is omitted due to generating mostly infeasible instances) and $\lambda_V = 1/20$. Overall, we generate 36 instances for each parameter combination, so we obtain 2736 `grid` instances.

**`geom`: Random Nearest Neighbor Graphs.** We use several parameters to generate MSP instances with 16 stages. The nodes in $G_1$ are $n \in \{1000, 2000, 5000\}$ randomly chosen real-valued points uniformly distributed over the unit square $[0, 1]^2$. We obtain the node position for each $G_{i+1}$ from $G_i$ by moving each node independently in a random direction chosen uniformly from $[-\frac{\varrho}{n}, \frac{\varrho}{n}]^2$ with $\varrho \in \{0, 1, 5\}$. In every stage, for each node we add an edge to its $k \in \{5, 10, 25, 50\}$ nearest neighbors (according to Euclidean distances). To allow for multiple shortest paths per stage, we consider two different weight functions: unit weights and exponential weights. The latter are generated by $2^{\lceil \log_2 100d \rceil}$, where $d$ is the Euclidean distance. This mapping partitions the otherwise very diverse edge weights into buckets of (exponentially) similar weights. Due to the factor 100 (and that we have a nearest neighbor graph) we mostly observe weights in $\{1, 2, 4, 8, 16\}$.

We consider these graphs with the *edge deletion* (with $\lambda_E \in \{1/2, 1/20\}$, omitting $\lambda_E = 1/2$ for $k = 5$ and $\lambda_E = 1/20$ for $(k, n) = (50, 5000)$ due to triviality) and the *weight scaling* modifications. We do not use *node deletion* here, as these modifications (even for small

non-trivial values of $\lambda_V$) resulted in mostly trivial instances (especially many infeasible ones). However, unless $\varrho = 0$, we also consider the instances without any further modifications since the random walks of the nodes already establish differences between the stages.

Overall, we generate 4 instances for each of the 240 parameter combinations with a unique query selected according to the scheme described in Section 3.2. We obtain 960 `geom` instances overall.

**`hybr`: Road Networks.** We use the undirected variants of the popular real-world `roadNet` data set [25], namely the road networks of California (`CA`), Pennsylvania (`PA`), and Texas (`TX`) as three distinct underlying graphs. As the original data set does not contain any temporal information, we use the three modification variants to obtain multistage instances, with $\lambda_E \in \{1/10, 1/20, 1/100\}$ and $\lambda_V \in \{1/20, 1/100\}$. In contrast to the artificial graphs, we observe that preprocessing the underlying graph w.r.t. a query yields dramatically smaller graphs (see Table 1). Thus, we performed the stage-wise modifications *after* preprocessing (i.e., we first choose a random query as described in Section 3.2, then preprocess, modify and check feasibility), in order to guarantee that the modifications are significant within the stages. The benchmark set `hybr` consists of overall 360 multistage instances with 4 stages and a unique query each.

**`real`: Communication Networks.** Many real-world graph data sets with timestamped edges are email data sets [22, 23], where nodes represent people and an edge indicates a message exchange at the indicated times. We use the data sets as provided by the Konect graph collection [23], but consider edges to be undirected. Here, timestamps are given with a relatively high resolution ranging between 1 and 100 seconds, meaning that only very few events happen exactly at the same time. To generate stages with a non-trivial number of edges, we have to decrease the temporal resolution, i.e., we generate stages by accumulating all events that occur during some time window. If we choose too large time windows, the stages become too dense and yield only very short shortest paths. On the other hand, if we have too many stages, there are typically no feasible non-trivial queries possible. We thus pick time window sizes that yield interesting graphs, but restrict ourselves to 2 or 8 consecutive stages. The queries are selected as described in Section 3.2.

- `enron` [22,23]: Email communication between employees of the energy corporation *Enron*. We use time window sizes of 168 hours (a week) and 744 hours (a month). To avoid too sparse (or obviously mislabeled) data, we only consider timestamps between May 27, 1998 and Feb 04, 2004 (a span of 297 weeks).
- `dnc-email` [23]: Email communication between members of the US Democratic National Committee. We use time window sizes of 24 and 48 hours. Here, we consider timestamps between Sep 16 2013 and May 25 2016 (a span of 140 weeks).

The benchmark set `real` consists of 80 2-stage ($66 \times$ `enron`, $14 \times$ `dnc-email`) and 20 8-stage instances ($14 \times$ `enron`, $6 \times$ `dnc-email`) that are selected as those instances with the lowest triviality score, which is the sum over the values $10 \cdot \mathbb{1}[t_S = 1 \vee t_L = 1] + t_S \cdot t_L$ for the considered transitions.

We also conducted the same process on human contact data sets [12, 21], where a timestamped edge indicates a measurement of physical proximity at the given point in time. However, the resulting graphs had either a very low diameter (3 or even lower, rendering MSP essentially trivial) if the time windows were too wide, or were highly disconnected if the time windows were too narrow. There was no sweet spot between these effects and thus these instances are not included. We also note that research tells us that autonomous systems and similar networks typically experience shrinking diameters over time [24], and are thus not well-suited to yield non-trivial MSP instances.

# 4 Algorithms

## 4.1 Exact Solutions

To compute exact MSP solutions, we propose a straight-forward integer linear programming (ILP) formulation. The preprocessing routine gives us the length $L_i$ of any shortest $s$-$t$-path in $G_i$, for each $i \in [\tau]$. Furthermore, it lets us define $\delta_i^+(v) \subseteq E_i$ ($\delta_i^-(v) \subseteq E_i$) as the edges that enter (leave) node $v$ when used in a shortest $s$-$t$-path. This allows us to use a directed flow formulation for assuring the path property.

For each stage $i \in [\tau]$, the binary variable $x_i(e)$ indicates whether edge $e \in E_i$ is in $P_i$. Constraints (1) ensure that each $P_i$ is an $s$-$t$-path, constraint (2) forces $P_i$ to be of shortest length. For each transition $(G_i, G_{i+1})$ the (de facto binary) variable $z_i(e)$ indicates – due to constraints (3) and (4) and the objective function – whether edge $e \in E_i \cap E_{i+1}$ is in $P_i \cap P_{i+1}$. Thus, the objective function maximizes the transition quality.

$$\max \sum_{i\in[\tau-1]} \sum_{e\in E_i} z_i(e)$$

$$s.t. \sum_{e\in\delta^-(v)} x_i(e) - \sum_{e\in\delta^+(v)} x_i(e) = \mathbb{1}[v = s] - \mathbb{1}[v = t] \qquad \forall\, i \in [\tau],\ \forall\, v \in V \quad (1)$$

$$\sum_{e\in E_i} w_i(e) \cdot x_i(e) = L_i \qquad \forall\, i \in [\tau] \quad (2)$$

$$z_i(e) \le x_i(e) \qquad \forall\, i \in [\tau-1],\ \forall\, e \in E_i \cap E_{i+1} \quad (3)$$

$$z_i(e) \le x_{i+1}(e) \qquad \forall\, i \in [\tau-1],\ \forall\, e \in E_i \cap E_{i+1} \quad (4)$$

$$x_i(e) \in \{0, 1\} \qquad \forall\, i \in [\tau],\ \forall\, e \in E_i \quad (5)$$

## 4.2 Two-Stage Algorithms

In the following, we make extensive use of the auxiliary algorithm `prefPath`$(i, F)$. It finds, among all shortest $s$-$t$-paths in $G_i$, a shortest $s$-$t$-path with the maximum number of edges from $F$. It does so by computing Dijkstra's algorithm w.r.t. the edge weights of $E_i$ where the weight of the edges in $F \cap E_i$ is reduced by some small $\varepsilon$. See [7] for details, where it is presented as the *preficiency algorithm* for MSP.

We first present some algorithms for the 2-stage problem MSP$|_2$, as these are later used as black-box algorithms for general MSP.

**Greedy (`G`):** Computes a shortest $s$-$t$-path $P_1 \leftarrow$ `prefPath`$(1, E_2)$ in $G_1$ and, favoring this path, a shortest $s$-$t$-path $P_2 \leftarrow$ `prefPath`$(2, P_1)$ in $G_2$.

**Double Greedy (`Gd`):** Calls `G` twice independently: once as described above, then with the roles of the stages interchanged. The output is the solution with larger transition quality.

**Iterated Greedy (`Gi`):** Computes $(P_1, P_2)$ with `G` and then alternatingly reoptimizes $P_i \leftarrow$ `prefPath`$(i, P_{3-i})$ for $i = 1, 2$ until the transition quality does not improve anymore.

**Approximation (`A`):** This $(2\mu)^{-1/2}$-approximation algorithm from [7] iteratively computes candidate solutions (pairs of paths) and finally outputs the pair with largest transition quality. Let $Y := E_1 \cap E_2$ be the initial set of edges to be preferred. In each iteration $j$, a pair of paths $P_1^{(j)} \leftarrow$ `prefPath`$(1, Y)$ and $P_2^{(j)} \leftarrow$ `prefPath`$(2, P_1^{(j)})$ is computed as a new candidate solution, and we update the set of preferred edges to $Y \leftarrow Y \setminus P_1^{(j)}$. According to [7], the algorithm continues until eventually $Y = \varnothing$ (which is guaranteed to happen due to our preprocessing). Our implementation can furthermore correctly halt earlier if the current best transition quality matches the upper bound $|E_1 \cap$ `prefPath`$(2, E_1)|$.

**Double Approximation (`Ad`):** Similarly to how `Gd` doubles `G`, this variant calls `A` twice, the second time with the roles of the two stages interchanged. It outputs the solution with larger transition quality.

**Bounded Approximation (`A5`):** A variation of `A` that halts after the first 5 candidate solutions (or earlier if `A` halts earlier).

### 4.3 Multistage Algorithms

We consider two different polynomial-time approaches to find solutions if $\tau > 2$.

**Multistage Greedy (`M-G`):** After initializing $P_1 \leftarrow$ `prefPath`$(1, E_2)$, subsequent paths $P_i \leftarrow$ `prefPath`$(i, P_{i-1})$ are computed iteratively for $i = 2, ..., \tau$. Proceeding in the reversed direction, for each $i = \tau - 1, ..., 1$ the solution $P_i$ is updated to `prefPath`$(i, P_{i+1})$. This process is repeated alternatingly front to back and back to front as long as the transition quality increases. Note that `M-G` for $\tau = 2$ coincides with `Gi`.

**Multistage with black-box (`B-*`):** This algorithm from [7] uses any MSP$|_2$-algorithm `*` as a black-box. The latter is executed on each consecutive pair of stages. Using a linear-time dynamic programming approach, it computes a collection of non-adjacent transitions whose transition qualities sum to the largest number. If the individual (2-stage) transitions are computed using some $\alpha$-approximation, this routine yields an $\frac{\alpha}{2}$-approximation; in the case of `B-A` we thus obtain an $(8\mu)^{-1/2}$-approximation.

Improving over the description in [7] in practice, our implementation does not use arbitrary solutions for a stage that is neither optimized to the previous nor to the next stage. Instead, considering such a stage $G_i$, we set $P_i$ to either `prefPath`$(i, P_{i-1})$ or `prefPath`$(i, P_{i+1})$, depending on which path yields the better transition qualities in conjunction with the solution paths of its neighboring stages (both of which are naturally fixed by the dynamic programming). We evaluate the algorithm's performance using each of the 2-stage algorithms described above as a black-box.

## 5 Experiments

The different algorithmic variants differ mainly in their approach for solving two-stage sub-instances. Therefore it is natural to first investigate their performance on MSP$|_2$ instances separately. Thereafter, we consider the multistage instances with $\tau > 2$.

Given some instance and some algorithm `X`, the *gap* is the ratio $(\mathsf{opt} - \mathsf{heu})/\mathsf{opt}$ where heu is the objective value computed by `X` and opt the optimal objective value.

**Hard- and Software.** All computations were run on an Intel Xeon Gold 6134 with 3.2 GHz and 256 GB RAM running Debian 9. We limit each run to a single thread with a 10 minute time limit. Our C++ (gcc 8.3.0) code uses OGDF Dogwood [6] as a graph algorithms library; the code will become part of the next OGDF release. We use CPLEX 20.1 as our ILP solver.

### 5.1 MSP$|_2$

To obtain MSP$|_2$ instances, we simply use the first two stages of every instance of `grid`, `geom` and `hybr`, as well as the two-stage instances from `real` (which, by construction, are selected to have better non-triviality than a random stage pair in the 8-stage `real` instances). See Table 2 for some average key figures on the two-stage experiments.

Nearly all two-stage algorithms are able to find solutions for all MSP$|_2$ instances within the time limit, except for `ILP`, which hits the time limit on 1.1% of the instances. In particular, due to the high success rate of `ILP` (whose few fails are restricted to very large `grid` instances), allows us to understand how often the heuristics and approximation algorithms yield optimal solutions as well. For `grid` and `geom` instances, the running times behave as one would expect on average: The greedy variants are fastest, followed by the `A` versions. The exact `ILP` is slower than the greedy approaches by up to 3 orders of magnitude (and still up to 2 orders of magnitude compared to `A` and `Ad`); only for `hybr` it is only roughly 10-fold slower

**Table 2 Results for MSP$|_2$ experiments:** The instances successfully solved by `ILP` yield a subset of each benchmark set for which we now know the optimal solutions. The "solved optimally" columns for `ILP` give the mean size of the respective subsets relative to the overall size of the benchmark sets. For the other algorithms, the values in the "solved optimally" columns, as well as the various "gap" columns, are then always given w.r.t. to these subsets. The columns "avg. gap" and "avg. gap (¬opt)" give the mean observed gaps to the optima, where the latter is restricted to the instances not solved to optimality by the considered algorithm. We suppress the "gap" columns for `real`, since all algorithms solved all these instances to optimality.

| | time [ms] | | | | solved optimally | | | | avg. gap | | | avg. gap (¬opt) | | | max. gap | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | grid | geom | hybr | real | grid | geom | hybr | real | grid | geom | hybr | grid | geom | hybr | grid | geom | hybr |
| ILP | 46787 | 897 | 2947 | 148 | *98.9%* | *100%* | *100%* | *100%* | — | — | — | — | — | — | — | — | — |
| G | **20** | **3** | **185** | 1 | 48.3% | 95.0% | 91.4% | 100% | 3.0% | 0.8% | 0.1% | 5.9% | 15.4% | 0.9% | 43.6% | 37.5% | 7.4% |
| Gd | 41 | 5 | 316 | 1 | 56.2% | 98.5% | **99.2%** | 100% | 1.5% | 0.2% | 0.0% | 3.4% | 11.2% | **0.3%** | 23.7% | 22.2% | **0.4%** |
| Gi | 31 | 4 | 256 | 1 | **70.4%** | 96.9% | 98.6% | 100% | **1.0%** | 0.5% | 0.0% | **3.2%** | 15.3% | 0.7% | 28.5% | 33.3% | 1.9% |
| A | 972 | 17 | 361 | 1 | 48.9% | 96.5% | 91.4% | 100% | 2.6% | 0.5% | 0.1% | 5.1% | 13.0% | 0.9% | 27.4% | 23.1% | 6.0% |
| Ad | 1913 | 34 | 660 | 2 | 56.6% | **99.0%** | **99.2%** | 100% | 1.4% | **0.1%** | 0.0% | **3.2%** | **9.5%** | **0.3%** | **19.5%** | **16.7%** | **0.4%** |
| A5 | 46 | 4 | 346 | 1 | 48.9% | 96.2% | 91.4% | 100% | 2.6% | 0.5% | 0.1% | 5.2% | 13.1% | 0.9% | 27.4% | 25.0% | 6.0% |

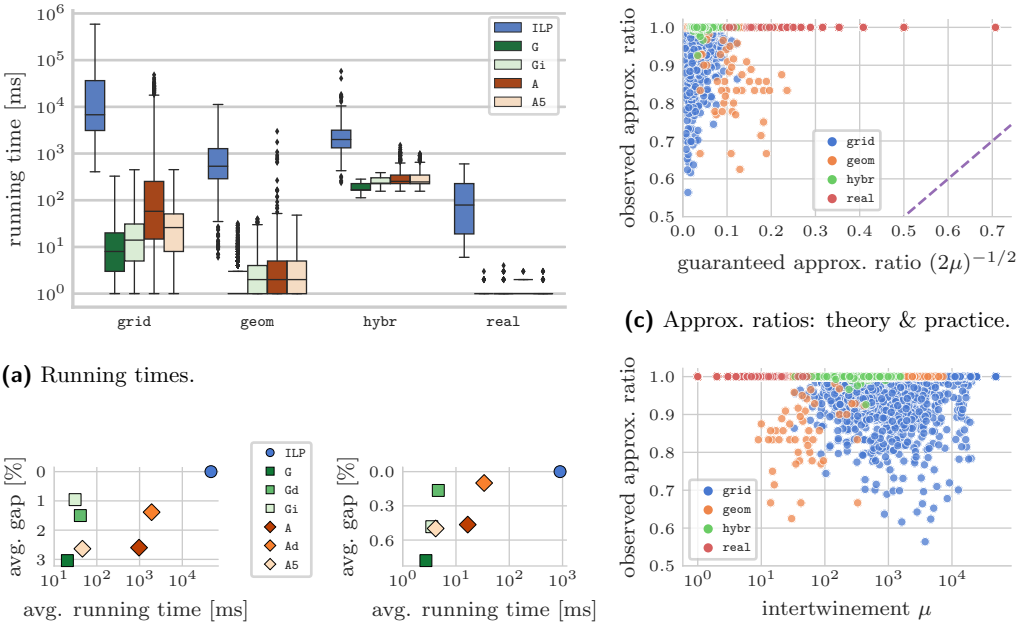
than the non-exact approaches. Naturally, `Gd` and `Ad` take about double the time of their basic counterparts. While `Gi` is slower than `G`, it is still faster than `Gd` on average: `G` and `Gd` require 2 and 4 calls to `prefPath`, respectively, but `Gi` typically terminates after the 3rd call, realizing that it cannot improve on the solution after the first two calls (i.e., the solution is identical to the one of `G`). Interestingly, `A5`'s running time is roughly comparable to that of `Gd`: it requires 2.64 iterations on average (and thus roughly 5 calls to `prefPath` on average, with a median of 2 calls), and does not suffer from outliers with a vast number of iterations as `A` does (see below). On the `hybr` benchmark set, `A` requires drastically fewer iterations than on `grid` and `geom`, and its running time becomes comparable to `A5` and thus not too far off from the greedy approaches. The running times on the `real` instances are negligibly small for all algorithms, so we refrain from analyzing them in detail.

However, as depicted in Figure 2a, the average running times do not tell the whole story. While most algorithms expose a rather predictable running time, the high variance in the running time of `A` is stunning: for many `grid` and `geom` instances, `A` spends a lot of time on later iterations that only yield candidate solutions with trivially small transition quality, but is unable to deduce that further iterations are futile.

For the following quality comparisons of the non-exact algorithms, we only consider instances with known optimal objective value (i.e. those that `ILP` could solve to proven optimality). The 2-stage `real` instances can all be solved to optimality by all algorithms. We conclude that they are, despite our best effort, still too trivial. Also the `hybr` and `geom` instances can typically be solved to optimality by most algorithms, with success rates of (clearly) above 90%. In contrast to this, the `grid` instances yield comparably hard instances for the heuristics (seemingly independent of the precise parameter choices). Note that this is also the only set where `ILP` sometimes fails to prove optimal solutions (for $(x, y) \in \{50\} \times \{500, 1000\}$). Interestingly, `Gi` finds the maximum number of optimal solutions (79.7%) overall.

Considering the average gaps, however, the difference in hardness between `grid` and `geom` seems to flip: even though many `grid` instances are not solved optimally, the observed gaps are relatively low, within one-digit percentages. In contrast to this, non-optimally solved `geom` instances typically yield gaps in the range of 10%–15% for all non-exact algorithms.

The two greedy variants `Gd` and `Gi` beat `G` w.r.t. the objective value on 20.7% and 32.5% of the instances, respectively. The average improvement over the initial greedy objective value in these cases is 30.2% and 21.6%, respectively.




**(a)** Running times.

**(c)** Approx. ratios: theory & practice.

**(b)** Trade-offs on `grid` (left) and `geom` (right).

**(d)** Observed approx. ratio vs. $\mu$.

**Figure 2 Visualizations for the MSP$|_2$ experiments.** **(a)** The boxes show the median and quartiles; the whiskers extend to the farthest data point within 1.5 times the interquartile range. **(b)–(d)** We show the average gaps on all instances with known optimum; a gap $g$ is equivalent to an observed approximation ratio of $1 - g$; the y-axes are arranged such that vertically higher data points represent solutions closer to the optimum.

For `A`, the average number of iterations (each iteration requiring two calls to `prefPath`) is 35.3. However, the actual output solution is already found after 1.2 iterations on average, generating an average computational overhead of 60.7% per instance for futile subsequent iterations. In fact, for 95.6% of the instances `A` already finds the optimal solution with the initial candidate solution (which is the same solution `G` finds). For nearly all instances (99.2%), `A` finds its output solution within the first 5 iterations, i.e., here `A5` outputs the same solution as `A`. Inversely, even if `A5` terminates earlier than `A`, this yields worse solutions only in 2.1% of those instances. Using `Ad` improves the objective value compared to `A` on 19.9% of the instances; the average improvement is 31.2% (coming at the cost of doubling the running time).

Algorithm `A` has an approximation ratio of $(2\mu)^{-1/2}$. As Figure 2c shows, `A` not only performs much better than the worst-case analysis suggests, but the correlation between the observed approximation ratio (which is $1 - g$ for gap $g$) and the intertwinement $\mu$ is just not very pronounced on our instances (as shown more clearly in Figure 2d). Clearly, the quite intricate instance structures necessary to yield weak approximations do typical not appear in practice (at least not in our benchmark sets).

Figure 2b shows the trade-off between solution quality (in terms of average gap over the instances solved by `ILP`) and required running time for all considered algorithms. We can conclude that the `ILP` should be preferred if running time is not an issue, and one of the two greedy approaches `Gi` or `Gd` in all other cases. The slightly better running time of `G` is typically not worth it due to the drop in quality. One could also make a case for `Ad` which, despite requiring much more time, sometimes finds slightly better solutions than the greedy variants.

**Table 3 Results for MSP ($\tau > 2$) experiments:** Columns are interpreted as in Table 2. Recall that the `grid`, `geom`, `hybr`, and `real` instances have 16, 16, 4, and 8 stages, respectively.

| | time [ms] | | | | solved optimally | | | | avg. gap (¬opt) | | | | max. gap | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | grid | geom | hybr | real | grid | geom | hybr | real | grid | geom | hybr | real | grid | geom | hybr | real |
| ILP | 165242 | 6453 | 4202 | 666 | *84.4%* | *100%* | *100%* | *100%* | — | — | — | — | — | — | — | — |
| B-G | **206** | 48 | **627** | 13 | 0.0% | 1.6% | 10.8% | 35.0% | 15.6% | 13.8% | 3.2% | 10.4% | 37.6% | 45.2% | 14.9% | 20.8% |
| B-Gd | 397 | 71 | 1023 | 19 | 0.0% | 1.7% | 11.4% | 35.0% | 14.7% | **13.4%** | 3.2% | 10.4% | 37.6% | 45.2% | **14.5%** | 20.8% |
| B-Gi | 338 | 58 | 839 | 16 | 0.0% | 1.6% | 11.1% | 35.0% | **14.1%** | 13.6% | 3.2% | 10.4% | 37.6% | 45.2% | **14.5%** | 20.8% |
| B-A | 13922 | 130 | 1185 | 13 | 0.0% | 1.5% | 10.8% | 35.0% | 15.5% | 13.6% | 3.2% | 10.4% | 37.6% | 45.2% | 14.9% | 20.8% |
| B-Ad | 19096 | 245 | 2107 | 22 | 0.0% | 1.6% | 11.4% | 35.0% | 14.6% | **13.4%** | 3.2% | 10.4% | 37.6% | 45.2% | **14.5%** | 20.8% |
| B-A5 | 572 | 68 | 1121 | 16 | 0.0% | 1.5% | 10.8% | 35.0% | 15.5% | 13.7% | 3.2% | 10.4% | 37.6% | 45.2% | 14.9% | 20.8% |
| M-G | 257 | **31** | 692 | **8** | **0.1%** | **2.1%** | **15.3%** | 0.0% | 16.8% | 23.5% | 3.2% | 23.0% | 45.1% | 68.8% | 23.7% | 42.9% |

## 5.2 MSP

Considering the true multistage instances, i.e., $\tau > 2$, we compare `M-G` and the various variants `B-{G,Gd,Gi,A,Ad,A5}`. Some key figures are presented in Table 3.

The `ILP`'s running times increase compared to the 2-stage scenarios, but not by as much as one might expect: compared to their 2-stage counterparts, the 16-stage `grid`, 16-stage `geom`, 4-stage `hybr`, and 8-stage `real` instances require roughly 3.5x, 7.2x, 1.4x, and 4.5x more time, respectively. Thus, while `ILP` is certainly a time-wise expensive algorithm, we still can solve nearly all multistage instances to proven optimality within the time limit: it only fails on roughly 1/6 of the `grid` instances. This still allows us to investigate the ability of the non-exact algorithms to find optimal solutions.

First we may consider their running times. Observe that all `B-*` variants first run their internal $\mathsf{MSP}|_2$ algorithm for $\tau - 1$ transitions. The subsequent dynamic programming over a sequence of only $\tau - 1$ integers requires negligible time compared to the various `prefPath`-calls. Hence, the running times of these algorithms are essentially the running times observed for their internal $\mathsf{MSP}|_2$ algorithms, scaled by the number of stage transitions. `B-Ad` is the only non-exact algorithm that (on 1.3% of the `grid` instances) runs into the time limit. The running time of `M-G` is very competitive and roughly comparable with the fastest `B-*` variant, namely `B-G`.

The most interesting finding is how seldom the heuristics and the approximation approach find optimal solutions. While they all do so in most of the cases for $\mathsf{MSP}|_2$, the situation changes drastically for $\tau > 2$: We may start with discussing the `B-*` variants, as they all yield essentially the same success rates: not a single multistage `grid` instance is solved to optimality (and only mediocre 19% and 11% of `geom` and `hybr`, respectively). Even for the previously too trivial `real` instances, the algorithms find optimal solutions only for roughly a third of the 8-stage instances. The reason for this consistent picture amongst all `B-*` variants is easy to see: generally, the solution quality for the individual 2-stage sub-problems is very similar. Their common ingredient, i.e., the selection of "good" non-adjacent transitions, is to blame for the weak performance. While it is theoretically sound to simply essentially "ignore" every second transition (while still retaining an approximation guarantee), this turns out to be abysmal in practice. In fact, we can see that this worst-case scenario is even (nearly) happening for some `geom` instances: despite the fact that half of the individual transitions are essentially optimal, we observe instances with an overall gap of 45.2% – very close to the worst case of 50%. Generally, this effect overshadows the influence of the precise selection of the black-box algorithm. Even when considering the gaps yielded by the non-optimal solutions, we only see slight deviations between the variants. Interestingly, the `hybr` instances allow generally lower gaps than the other benchmark sets.

Now, one may hope that the straight-forward but reasonable sounding heuristic `M-G` may fare better, but this is also hardly the case: it finds (only) two optimal solutions on `grid` instances and is slightly more successful than `B-*` on `geom` and `hybr`. For the `real` instances, however, is fails to find any optimal solution at all. Generally over all benchmark sets, its obtained gaps are weaker than those of `B-*`. In fact, on the `grid` instances its gaps can become close to 50% and for `geom` it even achieves a solution quality only 31.2% of the optimum (a gap of 68.8%).

Overall, we can see that no non-exact algorithm comes close to the optimal solution quality obtained by `ILP`, which is thus the probably best algorithmic choice – if time is not an issue. Otherwise, we would have to recommend the use of `B-G` or `M-G`, which are comparable in quality and running time. The other more expensive $\mathsf{MSP}|_2$ algorithms are not worth it when used within the `B-*` context on these instances.

## 6 Conclusion

In theory, the only known approximations for $\mathsf{MSP}|_2$ and general MSP (`A` and `B-A`) guarantee ratios of $(2\mu)^{-1/2}$ and $(8\mu)^{-1/2} = 1/2 \cdot (2\mu)^{-1/2}$, respectively, where `B-A` uses `A` internally and only causes an additional constant ratio of $1/2$. Thus, in the hunt for better (in particular constant) approximation ratios, it seems natural to focus on stronger approximations for $\mathsf{MSP}|_2$. However, our study shows that this is precisely *not* the interesting question when we want to obtain practically strong algorithms: $\mathsf{MSP}|_2$ is rather simple to solve in practice, the worst-case ratios of `A` are never met, and even simple greedy heuristics find close-to-optimal solutions. In contrast, the lifting from 2 to $\tau > 2$ stages is a central weak point which undermines the algorithms' success. Also, straight-forward alternative greedy strategies (`M-G`) do not work well. We therefore propose to focus on finding true multistage approximation routines, instead of relying on simple liftings from algorithms for few stages.

In [7], a more general version of `A` is presented that is applicable to all subgraph problems that are *preficient*, which roughly speaking means that they allow a routine along the lines of `prefPath`. Thus, all `G` variants (as well as `M-G`) can also be used there, and we wonder if they perform similarly strong for such other problems as they do for $\mathsf{MSP}|_2$ (MSP, respectively).

## References


1 Evripidis Bampis, Bruno Escoffier, and Alexander Kononov. LP-based algorithms for multistage minimization problems. In Christos Kaklamanis and Asaf Levin, editors, *Approximation and Online Algorithms*, Lecture Notes in Computer Science, pages 1–15. Springer International Publishing, 2021. doi:10.1007/978-3-030-80879-2_1.

2 Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th Paschos. Multistage matchings. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. ISSN: 1868-8969. doi:10.4230/LIPIcs.SWAT.2018.7.

3 Evripidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller. Online multistage subset maximization problems. *Algorithmica*, 83(8):2374–2399, 2021-08-01. doi:10.1007/s00453-021-00834-7.

4 Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. Multistage knapsack. *Journal of Computer and System Sciences*, 126:106–118, 2022-06-01. doi:10.1016/j.jcss.2022.01.002.

5 Béla Bollobás and Oliver Riordan. The diameter of a scale-free RandomGraph. *Combinatorica*, 24(1):5–34, 2004-01-01. doi:10.1007/s00493-004-0002-2.

6 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (OGDF). In Roberto Tamassia, editor, *Handbook on graph drawing and visualization*, pages 543–569. Chapman and Hall/CRC, 2013.

7 Markus Chimani, Niklas Troost, and Tilo Wiedera. A general approach to approximate multistage subgraph problems, 2021-07-06. doi:10.48550/arXiv.2107.02581.

8 Markus Chimani, Niklas Troost, and Tilo Wiedera. Approximating multistage matching problems. *Algorithmica*, 84(8):2135–2153, 2022-08-01. doi:10.1007/s00453-022-00951-x.

9 Fan Chung and Linyuan Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, 2001-05-01. doi:10.1006/aama.2001.0720.

10 Huanqing Cui, Ruixue Liu, Shaohua Xu, and Chuanai Zhou. DMGA: A distributed shortest path algorithm for multistage graph. *Scientific Programming*, 2021:e6639008, 2021-06-01. Publisher: Hindawi. doi:10.1155/2021/6639008.

11 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, 1959-12-01. doi:10.1007/BF01386390.

12 Nathan Eagle and Alex (Sandy) Pentland. Reality mining: sensing complex social systems. *Pers Ubiquit Comput*, 10(4):255–268, 2006-05-01. doi:10.1007/s00779-005-0046-3.

13 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 459–470. Springer, 2014. doi:10.1007/978-3-662-43951-7_39.

14 D. Eppstein, M. S. Paterson, and F. F. Yao. On nearest-neighbor graphs. *Discrete Comput Geom*, 17(3):263–282, 1997-04-01. doi:10.1007/PL00009293.

15 Till Fluschnik. A multistage view on 2-satisfiability. In Tiziana Calamoneri and Federico Corò, editors, *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 231–244. Springer International Publishing, 2021. doi:10.1007/978-3-030-75242-2_16.

16 Till Fluschnik, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. Multistage vertex cover. *Theory Comput Syst*, 66(2):454–483, 2022-04-01. doi:10.1007/s00224-022-10069-w.

17 Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. Multistage s-t path: Confronting similarity with dissimilarity in temporal graphs. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. ISSN: 1868-8969. doi:10.4230/LIPIcs.ISAAC.2020.43.

18 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 563–575. Springer, 2014. doi:10.1007/978-3-662-43948-7_47.

19 Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theoretical Computer Science*, 868:46–64, 2021-05-08. doi:10.1016/j.tcs.2021.04.002.

20 Johan Håstad. Some optimal inapproximability results. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 1–10. Association for Computing Machinery, 1997-05-04. doi:10.1145/258533.258536.

21 Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011-02-21. doi:10.1016/j.jtbi.2010.11.033.

22 Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Machine Learning: ECML 2004*, Lecture Notes in Computer Science, pages 217–226. Springer, 2004. doi:10.1007/978-3-540-30115-8_22.

**23** Jérôme Kunegis. KONECT: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350. Association for Computing Machinery, 2013-05-13. doi:10.1145/2487788.2488173.

**24** Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187. Association for Computing Machinery, 2005-08-21. doi:10.1145/1081870.1081893.

**25** Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, 2008-10-08. doi:10.48550/arXiv.0810.1355.

# A APPENDIX

**Table 4 Instance characteristics of `grid` graphs.** The columns are interpreted as in Table 1. Instance sets with "—" are not generated, while " ⁄ " indicates no known optimal solution. A mean hop-count value in italics is ignoring instances with no known optimum.

| `grid` | $x = 5$ | | | | $x = 10$ | | | | $x = 25$ | | | | $x = 50$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ |
| $n = 100$, $\lambda_E = 0.05$ | 96.0% | 95.3% | 349.8 | 104.5 | 91.4% | 90.6% | 911.9 | 108.1 | 96.3% | 96.1% | 3531.4 | *123.0* | 97.8% | 97.7% | 7771.4 | ⁄ |
| $n = 100$, $\lambda_E = 0.1$ | 97.7% | 96.8% | 245.0 | 108.2 | 81.5% | 79.4% | 395.9 | 109.8 | 83.1% | 82.3% | 2206.0 | *123.0* | 90.5% | 90.1% | 5802.2 | ⁄ |
| $n = 100$, $\lambda_E = 0.2$ | — | — | — | — | 84.6% | 80.2% | 198.4 | 117.1 | 60.6% | 57.4% | 455.1 | 124.5 | 69.7% | 68.0% | 2258.6 | *148.0* |
| $n = 100$, $\lambda_V = 0.05$ | 97.2% | 96.5% | 340.5 | 104.5 | 90.6% | 89.8% | 824.5 | 108.1 | 95.5% | 95.3% | 3204.2 | *123.0* | 97.6% | 97.6% | 7029.1 | ⁄ |
| $n = 100$, `scaling` | 99.9% | 99.8% | 515.8 | 103.0 | 100.0% | 100.0% | 1051.7 | 108.0 | 99.2% | 99.1% | 2383.9 | 123.0 | 100.0% | 100.0% | 4668.6 | 148.0 |
| $n = 200$, $\lambda_E = 0.05$ | 97.1% | 96.3% | 587.7 | 207.8 | 82.3% | 80.6% | 1003.3 | 209.4 | 83.3% | 82.7% | 5291.2 | ⁄ | 92.4% | 92.2% | 14171.8 | ⁄ |
| $n = 200$, $\lambda_E = 0.1$ | 98.3% | 97.2% | 405.3 | 215.5 | 83.9% | 80.7% | 551.4 | 215.3 | 60.3% | 58.5% | 1885.6 | *223.7* | 75.4% | 74.7% | 8791.3 | ⁄ |
| $n = 200$, $\lambda_E = 0.2$ | — | — | — | — | 89.3% | 84.2% | 306.9 | 231.7 | 60.4% | 54.8% | 449.1 | 234.3 | 44.6% | 42.2% | 1272.7 | 249.8 |
| $n = 200$, $\lambda_V = 0.05$ | 96.5% | 95.4% | 502.5 | 208.1 | 83.9% | 82.1% | 871.8 | 209.7 | 84.8% | 84.3% | 4763.4 | *223.0* | 91.8% | 91.6% | 12546.6 | ⁄ |
| $n = 200$, `scaling` | 100.0% | 100.0% | 1008.9 | 203.0 | 99.8% | 99.8% | 2078.2 | 208.0 | 100.0% | 100.0% | 5310.8 | 223.0 | 100.0% | 100.0% | 9824.1 | *248.0* |
| $n = 500$, $\lambda_E = 0.05$ | 98.7% | 98.0% | 1149.8 | 518.1 | 84.9% | 81.9% | 1451.2 | 517.6 | 57.7% | 56.1% | 3602.3 | *524.5* | 70.1% | 69.5% | 22019.8 | ⁄ |
| $n = 500$, $\lambda_E = 0.1$ | 99.3% | 98.6% | 873.9 | 538.1 | 90.5% | 86.5% | 908.7 | *534.4* | 60.5% | 56.1% | 1308.3 | 536.3 | 41.7% | 40.1% | 3725.3 | *550.3* |
| $n = 500$, $\lambda_E = 0.2$ | — | — | — | — | 93.7% | 88.8% | 642.1 | 575.1 | 67.4% | 58.2% | 582.6 | 571.6 | 43.4% | 37.1% | 738.7 | 577.1 |
| $n = 500$, $\lambda_V = 0.05$ | 98.7% | 98.0% | 1056.4 | 519.1 | 84.1% | 81.4% | 1244.6 | 518.1 | 55.5% | 53.8% | 3009.1 | *524.8* | 69.0% | 68.5% | 18871.9 | ⁄ |
| $n = 500$, `scaling` | 100.0% | 100.0% | 2505.7 | 503.0 | 100.0% | 100.0% | 5215.7 | 508.0 | 99.7% | 99.7% | 13234.9 | *523.0* | 100.0% | 100.0% | 26715.7 | ⁄ |
| $n = 1000$, $\lambda_E = 0.05$ | 99.0% | 98.3% | 1935.5 | *1035.9* | 90.4% | 87.5% | 2241.7 | *1032.8* | 58.4% | 55.0% | 3006.2 | *1034.9* | 41.7% | 40.6% | 9908.8 | ⁄ |
| $n = 1000$, $\lambda_E = 0.1$ | 99.6% | 99.0% | 1574.0 | *1074.9* | 93.8% | 90.2% | 1753.5 | *1065.8* | 67.5% | 60.3% | 1628.0 | *1064.5* | 43.6% | 39.0% | 2207.7 | *1070.5* |
| $n = 1000$, $\lambda_E = 0.2$ | — | — | — | — | 94.9% | 89.5% | 1103.1 | *1146.7* | 74.7% | 64.0% | 1019.0 | *1137.1* | 50.1% | 40.6% | 924.4 | *1137.2* |
| $n = 1000$, $\lambda_V = 0.05$ | 99.3% | 98.7% | 1808.1 | *1037.4* | 90.4% | 87.3% | 2200.1 | *1033.9* | 59.7% | 55.9% | 2793.2 | *1036.0* | 41.7% | 40.5% | 7904.4 | *1050.1* |
| $n = 1000$, `scaling` | 99.8% | 99.7% | 5105.9 | *1003.0* | 99.8% | 99.8% | 10082.5 | ⁄ | 99.8% | 99.7% | 25728.0 | ⁄ | 99.8% | 99.8% | 48757.2 | ⁄ |

**Table 5 Instance characteristics of `hybr` graphs.** We use the same notation as in Table 4.

| `hybr` | `CA` | | | | `PA` | | | | `TX` | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ |
| $\lambda_E = 0.01$ | 0.08% | 0.06% | 744.1 | 668.0 | 0.13% | 0.10% | 752.9 | 608.1 | 0.13% | 0.10% | 945.2 | 813.2 |
| $\lambda_E = 0.05$ | 0.12% | 0.09% | 472.6 | 763.8 | 0.19% | 0.15% | 474.1 | 648.5 | 0.20% | 0.16% | 510.8 | 857.3 |
| $\lambda_E = 0.1$ | 0.15% | 0.12% | 355.5 | 876.2 | 0.23% | 0.19% | 316.4 | 700.6 | 0.26% | 0.20% | 306.8 | 939.7 |
| $\lambda_V = 0.01$ | 0.08% | 0.06% | 684.5 | 677.1 | 0.14% | 0.11% | 792.2 | 626.0 | 0.13% | 0.10% | 896.0 | 816.1 |
| $\lambda_V = 0.05$ | 0.13% | 0.11% | 423.6 | 782.2 | 0.21% | 0.17% | 409.9 | 656.3 | 0.23% | 0.18% | 451.1 | 878.2 |
| `scaling` | 0.11% | 0.08% | 747.5 | 719.0 | 0.23% | 0.18% | 562.6 | 682.3 | 0.18% | 0.14% | 813.9 | 855.4 |

**Table 6 Instance characteristics of `geom` graphs.** We use the same notation as in Table 4. Weights are given either as unit weights (denoted as $1^x$) or exponential weights ($2^x$).

| `geom` | $k=5$ | | | | $k=10$ | | | | $k=25$ | | | | $k=50$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ | $\frac{n'}{n}$ | $\frac{m'}{m}$ | $\mu$ | $h$ |
| $n=1000$, $1^x$, $\varrho=1$, $\lambda_E=0$ | 19.5% | 10.3% | 142.0 | 26.8 | 13.6% | 6.2% | 258.5 | 16.7 | 13.9% | 4.5% | 444.2 | 9.2 | 8.5% | 1.8% | 444.2 | 6.0 |
| $n=1000$, $2^x$, $\varrho=1$, $\lambda_E=0$ | 10.1% | 4.9% | 55.8 | 33.2 | 7.3% | 2.2% | 66.8 | 22.1 | 5.4% | 0.7% | 34.2 | 13.6 | 4.4% | 0.3% | 46.5 | 8.1 |
| $n=1000$, $1^x$, $\varrho=5$, $\lambda_E=0$ | 24.2% | 12.0% | 70.8 | 26.7 | 22.8% | 10.2% | 144.8 | 16.8 | 22.5% | 8.7% | 394.0 | 9.3 | 12.3% | 2.5% | 307.8 | 6.0 |
| $n=1000$, $2^x$, $\varrho=5$, $\lambda_E=0$ | 16.1% | 7.3% | 33.2 | 29.2 | 13.9% | 4.6% | 37.0 | 22.0 | 10.3% | 1.9% | 34.0 | 14.4 | 8.0% | 0.8% | 38.0 | 7.7 |
| $n=1000$, $1^x$, $\varrho=0$, $\lambda_E=0.2$ | 19.5% | 11.1% | 43.8 | 29.1 | 16.0% | 7.0% | 84.8 | 17.9 | 16.0% | 4.9% | 193.2 | 10.1 | 11.2% | 2.3% | 316.2 | 6.0 |
| $n=1000$, $2^x$, $\varrho=0$, $\lambda_E=0.2$ | 13.8% | 7.4% | 30.2 | 33.0 | 9.8% | 3.1% | 29.0 | 22.0 | 9.3% | 1.6% | 49.2 | 15.4 | 5.5% | 0.5% | 23.2 | 8.0 |
| $n=1000$, $1^x$, $\varrho=1$, $\lambda_E=0.2$ | 21.0% | 11.9% | 41.0 | 29.3 | 18.6% | 8.4% | 76.0 | 17.0 | 16.2% | 5.3% | 210.2 | 9.6 | 12.2% | 2.4% | 271.8 | 6.0 |
| $n=1000$, $2^x$, $\varrho=1$, $\lambda_E=0.2$ | 15.4% | 8.1% | 28.2 | 31.7 | 12.4% | 4.0% | 28.2 | 22.3 | 8.3% | 1.3% | 30.8 | 14.2 | 5.7% | 0.5% | 33.5 | 7.9 |
| $n=1000$, $1^x$, $\varrho=5$, $\lambda_E=0.2$ | 24.4% | 11.6% | 29.2 | 28.5 | 22.9% | 9.8% | 74.0 | 16.9 | 19.7% | 6.3% | 161.8 | 9.9 | 16.2% | 4.2% | 337.8 | 6.1 |
| $n=1000$, $2^x$, $\varrho=5$, $\lambda_E=0.2$ | 20.8% | 9.1% | 22.2 | 32.0 | 15.7% | 4.9% | 18.2 | 21.5 | 12.4% | 2.1% | 18.0 | 14.8 | 6.7% | 0.5% | 15.8 | 7.6 |
| $n=1000$, $1^x$, $\varrho=0$, $\lambda_E=0.5$ | — | — | — | — | 22.2% | 10.0% | 19.8 | 20.8 | 26.1% | 8.1% | 55.0 | 11.1 | 16.8% | 4.5% | 124.5 | 6.6 |
| $n=1000$, $2^x$, $\varrho=0$, $\lambda_E=0.5$ | — | — | — | — | 15.8% | 5.5% | 11.2 | 24.7 | 10.9% | 1.9% | 9.8 | 13.8 | 8.0% | 0.8% | 10.5 | 8.6 |
| $n=1000$, $1^x$, $\varrho=1$, $\lambda_E=0.5$ | — | — | — | — | 26.8% | 10.6% | 17.8 | 20.7 | 20.6% | 5.6% | 51.0 | 10.6 | 21.5% | 4.6% | 92.0 | 6.4 |
| $n=1000$, $2^x$, $\varrho=1$, $\lambda_E=0.5$ | — | — | — | — | 16.2% | 5.4% | 9.8 | 23.8 | 10.6% | 2.0% | 12.8 | 13.5 | 8.5% | 0.9% | 11.5 | 8.4 |
| $n=1000$, $1^x$, $\varrho=5$, $\lambda_E=0.5$ | — | — | — | — | 24.2% | 9.3% | 23.5 | 18.9 | 24.0% | 6.9% | 47.2 | 10.2 | 20.0% | 4.6% | 104.8 | 6.3 |
| $n=1000$, $2^x$, $\varrho=5$, $\lambda_E=0.5$ | — | — | — | — | 19.0% | 5.7% | 10.2 | 23.8 | 11.9% | 2.2% | 9.5 | 13.6 | 8.9% | 0.7% | 8.8 | 8.0 |
| $n=1000$, $1^x$, $\varrho=0$, `scaling` | 23.8% | 12.1% | 105.2 | 30.0 | 29.4% | 11.9% | 142.8 | 19.6 | 26.8% | 8.1% | 233.8 | 10.8 | 20.2% | 4.2% | 444.5 | 6.5 |
| $n=1000$, $2^x$, $\varrho=0$, `scaling` | 19.0% | 8.6% | 48.8 | 34.7 | 20.1% | 5.9% | 46.2 | 23.0 | 20.4% | 2.8% | 30.5 | 16.1 | 9.2% | 0.7% | 44.5 | 9.2 |
| $n=1000$, $1^x$, $\varrho=1$, `scaling` | 28.3% | 14.4% | 83.8 | 28.5 | 34.9% | 14.2% | 102.2 | 18.7 | 32.5% | 10.0% | 356.8 | 10.3 | 25.6% | 6.2% | 703.8 | 6.7 |
| $n=1000$, $2^x$, $\varrho=1$, `scaling` | 28.9% | 12.7% | 48.8 | 35.8 | 23.2% | 6.4% | 42.2 | 25.6 | 20.2% | 2.8% | 26.2 | 16.5 | 12.3% | 1.0% | 47.2 | 8.8 |
| $n=1000$, $1^x$, $\varrho=5$, `scaling` | 31.9% | 14.2% | 50.5 | 28.9 | 41.6% | 15.6% | 66.0 | 18.0 | 37.0% | 12.2% | 230.2 | 10.1 | 30.5% | 6.2% | 191.0 | 6.5 |
| $n=1000$, $2^x$, $\varrho=5$, `scaling` | 28.1% | 10.8% | 24.5 | 34.8 | 24.9% | 6.7% | 25.8 | 23.5 | 23.7% | 3.2% | 20.0 | 15.7 | 15.0% | 1.2% | 23.2 | 9.3 |
| $n=2000$, $1^x$, $\varrho=1$, $\lambda_E=0$ | 7.9% | 4.0% | 126.5 | 38.0 | 12.1% | 4.9% | 366.5 | 24.4 | 11.6% | 3.9% | 921.8 | 13.9 | 11.3% | 2.9% | 1411.0 | 8.9 |
| $n=2000$, $2^x$, $\varrho=1$, $\lambda_E=0$ | 8.3% | 4.1% | 90.2 | 52.4 | 5.2% | 1.5% | 86.5 | 33.6 | 5.5% | 0.8% | 111.2 | 21.3 | 4.9% | 0.4% | 124.8 | 16.9 |
| $n=2000$, $1^x$, $\varrho=5$, $\lambda_E=0$ | 17.0% | 9.0% | 117.0 | 39.4 | 17.4% | 7.7% | 251.0 | 24.9 | 22.8% | 8.3% | 983.8 | 14.7 | 10.0% | 2.0% | 559.5 | 9.0 |
| $n=2000$, $2^x$, $\varrho=5$, $\lambda_E=0$ | 13.6% | 6.4% | 60.5 | 50.2 | 12.9% | 4.2% | 64.0 | 34.4 | 10.8% | 2.1% | 67.5 | 21.2 | 8.9% | 1.0% | 92.0 | 17.7 |
| $n=2000$, $1^x$, $\varrho=0$, $\lambda_E=0.2$ | 11.8% | 6.8% | 45.0 | 39.8 | 18.6% | 8.2% | 232.5 | 26.4 | 12.3% | 4.3% | 270.8 | 14.5 | 14.1% | 3.9% | 968.2 | 9.5 |
| $n=2000$, $2^x$, $\varrho=0$, $\lambda_E=0.2$ | 13.9% | 7.5% | 60.5 | 56.1 | 10.8% | 3.4% | 64.2 | 35.0 | 7.2% | 1.1% | 60.8 | 21.0 | 7.1% | 0.6% | 55.5 | 17.2 |
| $n=2000$, $1^x$, $\varrho=1$, $\lambda_E=0.2$ | 16.2% | 9.1% | 53.8 | 41.3 | 16.4% | 7.6% | 126.2 | 26.2 | 19.7% | 6.9% | 365.5 | 14.9 | 17.6% | 4.6% | 337.8 | 9.5 |
| $n=2000$, $2^x$, $\varrho=1$, $\lambda_E=0.2$ | 12.8% | 7.1% | 47.5 | 50.2 | 10.7% | 3.6% | 59.5 | 36.2 | 7.8% | 1.3% | 45.8 | 20.8 | 6.1% | 0.6% | 35.0 | 16.6 |
| $n=2000$, $1^x$, $\varrho=5$, $\lambda_E=0.2$ | 20.2% | 9.9% | 42.0 | 40.9 | 19.9% | 8.3% | 89.2 | 26.4 | 21.0% | 7.2% | 371.5 | 14.9 | 17.0% | 4.4% | 750.5 | 9.0 |
| $n=2000$, $2^x$, $\varrho=5$, $\lambda_E=0.2$ | 15.4% | 7.1% | 35.8 | 51.3 | 14.1% | 4.8% | 32.2 | 35.0 | 10.6% | 1.8% | 34.5 | 19.7 | 9.1% | 0.9% | 40.5 | 15.8 |
| $n=2000$, $1^x$, $\varrho=0$, $\lambda_E=0.5$ | — | — | — | — | 19.6% | 7.9% | 26.5 | 28.7 | 21.3% | 6.7% | 73.5 | 15.9 | 22.6% | 6.3% | 204.0 | 9.9 |
| $n=2000$, $2^x$, $\varrho=0$, $\lambda_E=0.5$ | — | — | — | — | 14.7% | 4.8% | 13.2 | 33.5 | 11.0% | 1.7% | 14.0 | 21.0 | 7.8% | 0.7% | 16.0 | 16.2 |
| $n=2000$, $1^x$, $\varrho=1$, $\lambda_E=0.5$ | — | — | — | — | 20.4% | 8.4% | 31.2 | 27.8 | 18.6% | 5.7% | 72.0 | 15.0 | 21.8% | 5.7% | 245.5 | 10.2 |
| $n=2000$, $2^x$, $\varrho=1$, $\lambda_E=0.5$ | — | — | — | — | 12.6% | 4.4% | 17.0 | 33.2 | 10.9% | 1.8% | 15.5 | 21.8 | 8.0% | 0.7% | 16.2 | 16.0 |
| $n=2000$, $1^x$, $\varrho=5$, $\lambda_E=0.5$ | — | — | — | — | 20.0% | 7.1% | 28.0 | 27.1 | 20.6% | 6.6% | 85.8 | 15.0 | 17.8% | 4.6% | 210.5 | 9.6 |
| $n=2000$, $2^x$, $\varrho=5$, $\lambda_E=0.5$ | — | — | — | — | 15.1% | 4.7% | 13.5 | 32.6 | 11.2% | 1.8% | 12.8 | 21.3 | 9.1% | 0.9% | 11.5 | 14.5 |
| $n=2000$, $1^x$, $\varrho=0$, `scaling` | 19.4% | 10.0% | 129.2 | 43.3 | 27.7% | 11.0% | 188.8 | 28.9 | 33.0% | 9.9% | 351.8 | 16.2 | 31.3% | 8.2% | 753.0 | 10.2 |
| $n=2000$, $2^x$, $\varrho=0$, `scaling` | 20.5% | 9.2% | 85.8 | 55.6 | 19.8% | 5.5% | 77.5 | 37.1 | 17.6% | 2.3% | 159.2 | 24.8 | 13.8% | 1.0% | 42.8 | 17.6 |
| $n=2000$, $1^x$, $\varrho=1$, `scaling` | 19.4% | 9.9% | 154.8 | 43.5 | 24.6% | 9.4% | 208.8 | 27.7 | 29.2% | 8.9% | 596.2 | 16.4 | 26.9% | 6.1% | 1041.5 | 10.8 |
| $n=2000$, $2^x$, $\varrho=1$, `scaling` | 18.8% | 8.9% | 80.8 | 57.1 | 23.8% | 6.6% | 76.5 | 36.8 | 20.4% | 2.8% | 71.2 | 23.3 | 15.7% | 1.1% | 70.5 | 18.1 |
| $n=2000$, $1^x$, $\varrho=5$, `scaling` | 30.4% | 13.0% | 88.0 | 41.9 | 36.2% | 12.8% | 119.0 | 28.6 | 35.6% | 9.7% | 299.8 | 16.2 | 34.3% | 7.6% | 355.2 | 10.8 |
| $n=2000$, $2^x$, $\varrho=5$, `scaling` | 28.5% | 11.0% | 46.5 | 53.9 | 26.6% | 7.0% | 40.2 | 36.0 | 19.7% | 2.8% | 33.8 | 22.2 | 16.6% | 1.4% | 33.2 | 17.3 |
| $n=5000$, $1^x$, $\varrho=1$, $\lambda_E=0$ | 5.9% | 3.0% | 206.5 | 63.7 | 7.7% | 3.3% | 455.0 | 39.7 | 13.2% | 4.5% | 2153.2 | 25.1 | 9.5% | 2.4% | 2729.5 | 16.3 |
| $n=5000$, $2^x$, $\varrho=1$, $\lambda_E=0$ | 5.9% | 3.0% | 214.0 | 80.2 | 7.2% | 2.6% | 393.2 | 66.1 | 6.0% | 1.1% | 401.8 | 53.8 | 4.3% | 0.4% | 283.5 | 24.3 |
| $n=5000$, $1^x$, $\varrho=5$, $\lambda_E=0$ | 10.2% | 5.3% | 188.2 | 64.0 | 14.5% | 6.2% | 392.5 | 41.3 | 17.1% | 6.2% | 1263.8 | 24.2 | 14.6% | 4.0% | 2787.8 | 16.7 |
| $n=5000$, $2^x$, $\varrho=5$, $\lambda_E=0$ | 9.6% | 4.9% | 160.0 | 75.8 | 12.3% | 4.9% | 263.0 | 67.1 | 8.6% | 1.9% | 294.0 | 46.3 | 8.5% | 1.1% | 284.8 | 22.7 |
| $n=5000$, $1^x$, $\varrho=0$, $\lambda_E=0.2$ | 14.1% | 7.8% | 94.5 | 69.7 | 13.0% | 5.6% | 229.0 | 44.0 | 14.2% | 4.9% | 621.0 | 24.2 | — | — | — | — |
| $n=5000$, $2^x$, $\varrho=0$, $\lambda_E=0.2$ | 10.8% | 6.2% | 98.8 | 80.5 | 8.0% | 3.2% | 138.5 | 62.0 | 7.2% | 1.4% | 193.2 | 45.9 | — | — | — | — |
| $n=5000$, $1^x$, $\varrho=1$, $\lambda_E=0.2$ | 11.7% | 6.3% | 70.8 | 64.0 | 12.6% | 5.5% | 194.0 | 41.9 | 13.4% | 4.6% | 575.8 | 24.4 | — | — | — | — |
| $n=5000$, $2^x$, $\varrho=1$, $\lambda_E=0.2$ | 11.7% | 6.4% | 82.0 | 81.6 | 11.5% | 4.4% | 160.0 | 62.5 | 9.6% | 2.0% | 190.0 | 45.2 | — | — | — | — |
| $n=5000$, $1^x$, $\varrho=5$, $\lambda_E=0.2$ | 12.8% | 6.5% | 56.8 | 65.9 | 14.7% | 6.2% | 164.2 | 41.8 | 16.2% | 5.9% | 549.5 | 24.0 | — | — | — | — |
| $n=5000$, $2^x$, $\varrho=5$, $\lambda_E=0.2$ | 12.9% | 6.4% | 69.2 | 79.1 | 13.1% | 4.9% | 121.2 | 63.4 | 11.5% | 2.4% | 133.0 | 46.2 | — | — | — | — |
| $n=5000$, $1^x$, $\varrho=0$, $\lambda_E=0.5$ | — | — | — | — | 14.1% | 5.7% | 48.2 | 45.6 | 17.9% | 5.2% | 146.8 | 25.3 | 18.7% | 5.0% | 509.8 | 17.3 |
| $n=5000$, $2^x$, $\varrho=0$, $\lambda_E=0.5$ | — | — | — | — | 12.5% | 4.6% | 30.8 | 60.4 | 10.1% | 2.0% | 43.8 | 40.1 | 8.2% | 1.0% | 36.8 | 21.3 |
| $n=5000$, $1^x$, $\varrho=1$, $\lambda_E=0.5$ | — | — | — | — | 14.9% | 5.8% | 47.8 | 46.9 | 19.1% | 5.5% | 154.5 | 26.6 | 17.8% | 4.2% | 436.0 | 17.0 |
| $n=5000$, $2^x$, $\varrho=1$, $\lambda_E=0.5$ | — | — | — | — | 12.4% | 4.4% | 25.8 | 62.2 | 11.2% | 2.2% | 38.2 | 40.5 | 9.0% | 1.0% | 45.2 | 22.1 |
| $n=5000$, $1^x$, $\varrho=5$, $\lambda_E=0.5$ | — | — | — | — | 17.4% | 6.1% | 34.5 | 47.5 | 18.2% | 5.2% | 108.0 | 25.8 | 18.2% | 3.9% | 253.5 | 17.8 |
| $n=5000$, $2^x$, $\varrho=5$, $\lambda_E=0.5$ | — | — | — | — | 12.9% | 4.4% | 29.2 | 61.8 | 12.5% | 2.5% | 47.5 | 40.5 | 9.2% | 1.0% | 34.8 | 21.6 |
| $n=5000$, $1^x$, $\varrho=0$, `scaling` | 14.1% | 7.0% | 219.2 | 69.8 | 19.6% | 7.5% | 417.0 | 46.1 | 21.1% | 5.9% | 1878.2 | 26.1 | 25.8% | 6.2% | 2086.5 | 18.0 |
| $n=5000$, $2^x$, $\varrho=0$, `scaling` | 16.4% | 8.2% | 152.5 | 83.1 | 20.5% | 6.8% | 224.5 | 69.5 | 17.8% | 2.9% | 274.8 | 48.4 | 14.0% | 1.3% | 207.8 | 24.5 |
| $n=5000$, $1^x$, $\varrho=1$, `scaling` | 20.2% | 10.3% | 210.5 | 69.8 | 22.3% | 9.2% | 606.8 | 44.3 | 26.0% | 7.5% | 595.5 | 26.3 | 25.3% | 7.3% | 4169.2 | 17.9 |
| $n=5000$, $2^x$, $\varrho=1$, `scaling` | 17.8% | 8.8% | 186.8 | 86.6 | 19.1% | 6.3% | 272.5 | 67.1 | 18.9% | 3.2% | 328.5 | 48.9 | 16.3% | 1.5% | 72.2 | 24.5 |
| $n=5000$, $1^x$, $\varrho=5$, `scaling` | 20.9% | 9.3% | 137.2 | 67.3 | 25.5% | 9.6% | 273.5 | 46.3 | 25.9% | 7.2% | 869.2 | 26.5 | 32.4% | 8.8% | 1726.5 | 18.3 |
| $n=5000$, $2^x$, $\varrho=5$, `scaling` | 23.6% | 10.6% | 115.8 | 84.9 | 24.2% | 7.7% | 171.5 | 67.5 | 24.9% | 4.4% | 249.2 | 50.8 | 18.3% | 1.8% | 144.2 | 24.2 |